geemap Documentation

Release 0.20.4

Qiusheng Wu

Mar 27, 2023

Contents:

CHAPTER 1

geemap

A Python package for interactive mapping with Google Earth Engine, ipyleaflet, and ipywidgets.

- GitHub repo: https://github.com/giswqs/geemap
- Documentation: https://geemap.org
- PyPI: https://pypi.org/project/geemap/
- Conda-forge: https://anaconda.org/conda-forge/geemap
- 360+ GEE notebook examples: https://github.com/giswqs/earthengine-py-notebooks
- GEE Tutorials on YouTube: https://youtube.com/@giswqs
- Free software: MIT license

Acknowledgment: This material is based upon work supported by the National Aeronautics and Space Administration (NASA) under Grant No. 80NSSC22K1742 issued through the Open Source Tools, Frameworks, and Libraries 2020 Program.

Contents

1.1 Introduction

Geemap is a Python package for interactive mapping with Google Earth Engine (GEE), which is a cloud computing platform with a multi-petabyte catalog of satellite imagery and geospatial datasets. During the past few years, GEE has become very popular in the geospatial community and it has empowered numerous environmental applications at local, regional, and global scales. GEE provides both JavaScript and Python APIs for making computational requests to the Earth Engine servers. Compared with the comprehensive documentation and interactive IDE (i.e., GEE JavaScript Code Editor) of the GEE JavaScript API, the GEE Python API has relatively little documentation and limited functionality for visualizing results interactively. The **geemap** Python package was created to fill this gap. It is built upon ipyleaflet and ipywidgets, and enables users to analyze and visualize Earth Engine datasets interactively within a Jupyter-based environment.

Geemap is intended for students and researchers, who would like to utilize the Python ecosystem of diverse libraries and tools to explore Google Earth Engine. It is also designed for existing GEE users who would like to transition from the GEE JavaScript API to Python API. The automated JavaScript-to-Python conversion module of the **geemap** package can greatly reduce the time needed to convert existing GEE JavaScripts to Python scripts and Jupyter notebooks.

For video tutorials and notebook examples, please visit https://geemap.org/tutorials. For complete documentation on geemap modules and methods, please visit https://geemap.org/geemap.

If you find geemap useful in your research, please consider citing the following papers to support my work. Thank you for your support.

- Wu, Q., (2020). geemap: A Python package for interactive mapping with Google Earth Engine. *The Journal of Open Source Software*, 5(51), 2305. https://doi.org/10.21105/joss.02305
- Wu, Q., Lane, C. R., Li, X., Zhao, K., Zhou, Y., Clinton, N., DeVries, B., Golden, H. E., & Lang, M. W. (2019). Integrating LiDAR data and multi-temporal aerial imagery to map wetland inundation dynamics using Google Earth Engine. *Remote Sensing of Environment*, 228, 1-13. https://doi.org/10.1016/j.rse.2019.04.015 (pdf | source code)

Check out the geemap workshop I presented at the GeoPython Conference 2021. This workshop gives a comprehensive introduction to the key features of geemap.

1.2 Features

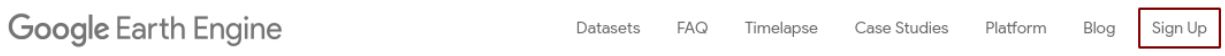
Below is a partial list of features available for the geemap package. Please check the examples page for notebook examples, GIF animations, and video tutorials.

- Convert Earth Engine JavaScripts to Python scripts and Jupyter notebooks.
- Display Earth Engine data layers for interactive mapping.
- Support Earth Engine JavaScript API-styled functions in Python, such as *Map.addLayer()*, *Map.setCenter()*, *Map.centerObject()*, *Map.setOptions()*.
- Create split-panel maps with Earth Engine data.
- Retrieve Earth Engine data interactively using the Inspector Tool.
- Interactive plotting of Earth Engine data by simply clicking on the map.
- Convert data format between GeoJSON and Earth Engine.
- Use drawing tools to interact with Earth Engine data.
- Use shapefiles with Earth Engine without having to upload data to one's GEE account.
- Export Earth Engine FeatureCollection to other formats (i.e., shp, csv, json, kml, kmz).
- Export Earth Engine Image and ImageCollection as GeoTIFF.

- Extract pixels from an Earth Engine Image into a 3D numpy array.
- Calculate zonal statistics by group.
- Add a customized legend for Earth Engine data.
- Convert Earth Engine JavaScripts to Python code directly within Jupyter notebook.
- Add animated text to GIF images generated from Earth Engine data.
- Add colorbar and images to GIF animations generated from Earth Engine data.
- Create Landsat timelapse animations with animated text using Earth Engine.
- Search places and datasets from Earth Engine Data Catalog.
- Use timeseries inspector to visualize landscape changes over time.
- Export Earth Engine maps as HTML files and PNG images.
- Search Earth Engine API documentation within Jupyter notebooks.
- Import Earth Engine assets from personal account.
- Publish interactive GEE maps directly within Jupyter notebook.
- Add local raster datasets (e.g., GeoTIFF) to the map.
- Perform image classification and accuracy assessment.
- Extract pixel values interactively and export as shapefile and csv.

1.3 Installation

To use **geemap**, you must first sign up for a Google Earth Engine account.



Geemap is available on PyPI. To install **geemap**, run this command in your terminal:

```
pip install geemap
```

Geemap is also available on conda-forge. If you have Anaconda or Miniconda installed on your computer, you can create a conda Python environment to install geemap:

```
conda create -n gee python=3.9
conda activate gee
conda install geopandas
conda install mamba -c conda-forge
mamba install geemap localtileserver -c conda-forge
```

Optionally, you can install Jupyter notebook extensions, which can improve your productivity in the notebook environment. Some useful extensions include Table of Contents, Gist-it, Autopep8, Variable Inspector, etc. See this post for more information.

```
conda install jupyter_contrib_nbextensions -c conda-forge
```

If you have installed **geemap** before and want to upgrade to the latest version, you can run the following command in your terminal:

```
pip install -U geemap
```

If you use conda, you can update geemap to the latest version by running the following command in your terminal:

```
conda update -c conda-forge geemap
```

To install the development version from GitHub using Git, run the following command in your terminal:

```
pip install git+https://github.com/giswqs/geemap
```

To install the development version from GitHub directly within Jupyter notebook without using Git, run the following code:

```
import geemap
geemap.update_package()
```

To use geemap in a Docker container, check out the following docker containers with geemap installed.

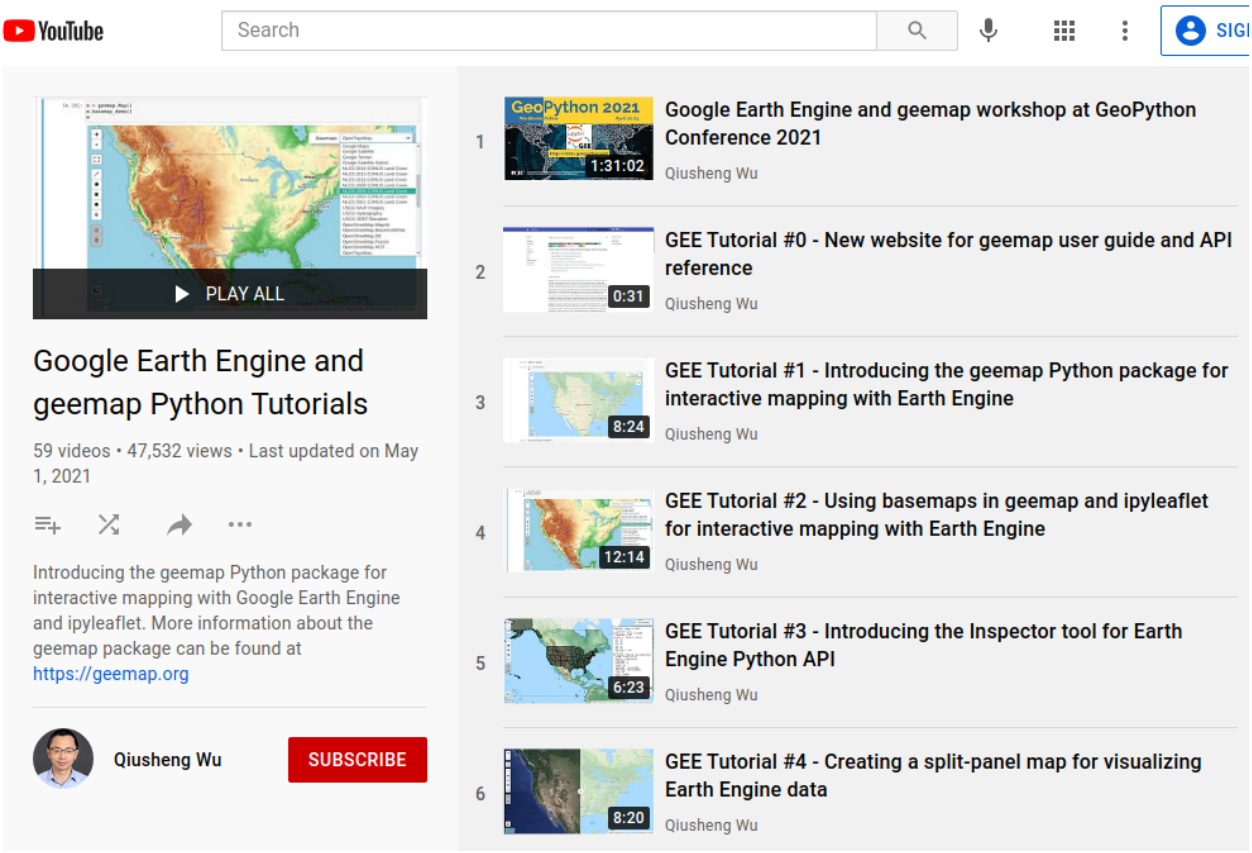
- gee-community/ee-jupyter-contrib
- bkavlak/geemap
- giswqs/geemap

To use geemap in a Docker container, check out ee-jupyter-contrib or this page.

1.4 Usage

Important note: A key difference between ipyleaflet and folium is that ipyleaflet is built upon ipywidgets and allows bidirectional communication between the front-end and the backend enabling the use of the map to capture user input, while folium is meant for displaying static data only (source). Note that Google Colab currently does not support ipyleaflet (source). Therefore, if you are using geemap with Google Colab, you should use import geemap.foliumap. If you are using geemap with binder or a local Jupyter notebook server, you can use import geemap, which provides more functionalities for capturing user input (e.g., mouse-clicking and moving).

More GEE Tutorials are available on my YouTube channel.



To create an ipyleaflet-based interactive map:

```
import geemap
Map = geemap.Map(center=[40,-100], zoom=4)
Map
```

To create a folium-based interactive map:

```
import geemap.foliumap as geemap
Map = geemap.Map(center=[40,-100], zoom=4)
Map
```

To add an Earth Engine data layer to the Map:

```
Map.addLayer(ee_object, vis_params, name, shown, opacity)
```

To center the map view at a given coordinates with the given zoom level:

```
Map.setCenter(lon, lat, zoom)
```

To center the map view around an Earth Engine object:

```
Map.centerObject(ee_object, zoom)
```

To add LayerControl to a folium-based Map:

```
Map.addLayerControl()
```

To add a minimap (overview) to an ipyleaflet-based Map:

```
Map.add_minimap()
```

To add additional basemaps to the Map:

```
Map.add_basemap('Esri Ocean')
Map.add_basemap('Esri National Geographic')
```

To add an XYZ tile layer to the Map:

```
url = 'https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}'
Map.add_tile_layer(url, name='Google Map', attribution='Google')
```

To add a WMS layer to the Map:

```
naip_url = 'https://services.nationalmap.gov/arcgis/services/USGSNAIPImagery/
↪ImageServer/WMSServer?'
Map.add_wms_layer(url=naip_url, layers='0', name='NAIP Imagery', format='image/png',␣
↪shown=True)
```

To convert a shapefile to Earth Engine object and add it to the Map:

```
ee_object = geemap.shp_to_ee(shp_file_path)
Map.addLayer(ee_object, {}, 'Layer name')
```

To convert a GeoJSON file to Earth Engine object and add it to the Map:

```
ee_object = geemap.geojson_to_ee(geojson_file_path)
Map.addLayer(ee_object, {}, 'Layer name')
```

To download an ee.FeatureCollection as a shapefile:

```
geemap.ee_to_csv(ee_object, filename, selectors)
```

To export an ee.FeatureCollection to other formats, including shp, csv, json, kml, and kmz:

```
geemap.ee_export_vector(ee_object, filename, selectors)
```

To export an ee.Image as a GeoTIFF file:

```
geemap.ee_export_image(ee_object, filename, scale, crs, region, file_per_band)
```

To export an ee.ImageCollection as GeoTIFF files:

```
geemap.ee_export_image_collection(ee_object, output, scale, crs, region, file_per_
↪band)
```

To extract pixels from an ee.Image into a 3D numpy array:

```
geemap.ee_to_numpy(ee_object, bands, region, properties, default_value)
```

To import a 2D or 3D numpy array to an ee.Image using a given base coordinate reference system (crs) and transform between projected coordinates and the base:

```
geemap.numpy_to_ee(np_array, crs, transform, transformWkt, band_names)
```

To import one or more variables from a netCDF file with a regular grid in EPSG:4326 to an ee.Image:

```
geemap.netcdf_to_ee(nc_file, var_names, band_names, lon='lon', lat='lat')
```

To calculate zonal statistics:

```
geemap.zonal_statistics(in_value_raster, in_zone_vector, out_file_path, statistics_
↪type='MEAN')
```

To calculate zonal statistics by group:

```
geemap.zonal_statistics_by_group(in_value_raster, in_zone_vector, out_file_path,␣
↪statistics_type='SUM')
```

To create a split-panel Map:

```
Map.split_map(left_layer='HYBRID', right_layer='ESRI')
```

To add a marker cluster to the Map:

```
Map.marker_cluster()
feature_collection = ee.FeatureCollection(Map.ee_markers)
```

To add a customized legend to the Map:

```
legend_dict = {
    'one': (0, 0, 0),
    'two': (255,255,0),
    'three': (127, 0, 127)
}
Map.add_legend(legend_title='Legend', legend_dict=legend_dict, position='bottomright')
Map.add_legend(builtin_legend='NLCD')
```

To download a GIF from an Earth Engine ImageCollection:

```
geemap.download_ee_video(tempCol, videoArgs, saved_gif)
```

To add animated text to an existing GIF image:

```
geemap.add_text_to_gif(in_gif, out_gif, xy=('5%', '5%'), text_sequence=1984, font_
↪size=30, font_color='#0000ff', duration=100)
```

To create a colorbar for an Earth Engine image:

```
palette = ['blue', 'purple', 'cyan', 'green', 'yellow', 'red']
create_colorbar(width=250, height=30, palette=palette, vertical=False,add_labels=True,
↪ font_size=20, labels=[-40, 35])
```

To create a Landsat timelapse animation and add it to the Map:

```
Map.add_landsat_ts_gif(label='Place name', start_year=1985, bands=['NIR', 'Red',
↪'Green'], frames_per_second=5)
```

To convert all GEE JavaScripts in a folder recursively to Python scripts:

```
from geemap.conversion import *
js_to_python_dir(in_dir, out_dir)
```

To convert all GEE Python scripts in a folder recursively to Jupyter notebooks:

```
from geemap.conversion import *
template_file = get_nb_template()
py_to_ipynb_dir(in_dir, template_file, out_dir)
```

To execute all Jupyter notebooks in a folder recursively and save output cells:

```
from geemap.conversion import *
execute_notebook_dir(in_dir)
```

To search Earth Engine API documentation with Jupyter notebooks:

```
import geemap
geemap.ee_search()
```

To publish an interactive GEE map with Jupyter notebooks:

```
Map.publish(name, headline, visibility)
```

To add a local raster dataset to the map:

```
Map.add_raster(image, bands, colormap, layer_name)
```

To get image basic properties:

```
geemap.image_props(image).getInfo()
```

To get image descriptive statistics:

```
geemap.image_stats(image, region, scale)
```

To remove all user-drawn geometries:

```
geemap.remove_drawn_features()
```

To extract pixel values based on user-drawn geometries:

```
geemap.extract_values_to_points(out_shp)
```

To load a Cloud Optimized GeoTIFF as an ee.Image:

```
image = geemap.load_GeoTIFF(URL)
```

To load a list of Cloud Optimized GeoTIFFs as an ee.ImageCollection:

```
collection = geemap.load_GeoTIFFs(URLs)
```

1.5 Examples

The following examples require the geemap package, which can be installed using `pip install geemap`. Check the *Installation* section for more information. More examples can be found at another repo: A collection of 300+ Jupyter Python notebook examples for using Google Earth Engine with interactive mapping.

- *Converting GEE JavaScripts to Python scripts and Jupyter notebooks*
- *Interactive mapping using GEE Python API and geemap*

1.5.1 Converting GEE JavaScripts to Python scripts and Jupyter notebooks

Launch an interactive notebook with **Google Colab**. Keep in mind that the conversion might not always work perfectly. Additional manual changes might still be needed. `ui` and `chart` are not supported. The source code for this automated conversion module can be found at conversion.py.

```
import os
from geemap.conversion import *

# Create a temporary working directory
work_dir = os.path.join(os.path.expanduser('~'), 'geemap')
# Get Earth Engine JavaScript examples. There are five examples in the geemap package␣
↪folder.
# Change js_dir to your own folder containing your Earth Engine JavaScripts, such as␣
↪js_dir = '/path/to/your/js/folder'
js_dir = get_js_examples(out_dir=work_dir)

# Convert all Earth Engine JavaScripts in a folder recursively to Python scripts.
js_to_python_dir(in_dir=js_dir, out_dir=js_dir, use_qgis=True)
print("Python scripts saved at: {}".format(js_dir))

# Convert all Earth Engine Python scripts in a folder recursively to Jupyter␣
↪notebooks.
nb_template = get_nb_template()  # Get the notebook template from the package folder.
py_to_ipynb_dir(js_dir, nb_template)

# Execute all Jupyter notebooks in a folder recursively and save the output cells.
execute_notebook_dir(in_dir=js_dir)
```

1.5.2 Interactive mapping using GEE Python API and geemap

Launch an interactive notebook with **Google Colab**. Note that **Google Colab** currently does not support ipyleaflet. Therefore, you should use `import geemap.foliumap` instead of `import geemap`.

```
# Installs geemap package
import subprocess

try:
        import geemap
except ImportError:
        print('geemap package not installed. Installing ...')
        subprocess.check_call(["python", '-m', 'pip', 'install', 'geemap'])

# Checks whether this notebook is running on Google Colab
try:
        import google.colab
        import geemap.foliumap as emap
except:
        import geemap as emap

# Authenticates and initializes Earth Engine
```

(continues on next page)

(continued from previous page)

```
import ee

try:
        ee.Initialize()
except Exception as e:
        ee.Authenticate()
        ee.Initialize()

# Creates an interactive map
Map = emap.Map(center=[40,-100], zoom=4)

# Adds Earth Engine dataset
image = ee.Image('USGS/SRTMGL1_003')

# Sets visualization parameters.
vis_params = {
        'min': 0,
        'max': 4000,
        'palette': ['006633', 'E5FFCC', '662A00', 'D8D8D8', 'F5F5F5']}

# Prints the elevation of Mount Everest.
xy = ee.Geometry.Point([86.9250, 27.9881])
elev = image.sample(xy, 30).first().get('elevation').getInfo()
print('Mount Everest elevation (m):', elev)

# Adds Earth Engine layers to Map
Map.addLayer(image, vis_params, 'SRTM DEM', True, 0.5)
Map.addLayer(xy, {'color': 'red'}, 'Mount Everest')
Map.setCenter(100, 40, 4)
# Map.centerObject(xy, 13)

# Display the Map
Map.addLayerControl()
Map
```

1.6 Dependencies

- bqplot
- colour
- earthengine-api
- folium
- geeadd
- geocoder
- ipyfilechooser
- ipyleaflet
- ipynb-py-convert
- ipytree

- ipywidgets
- mss
- pillow
- pyshp
- xarray-leaflet

1.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.7.1 Report Bugs

Report bugs at https://github.com/giswqs/geemap/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

1.7.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

1.7.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

1.7.4 Write Documentation

geemap could always use more documentation, whether as part of the official geemap docs, in docstrings, or even on the web in blog posts, articles, and such.

1.7.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/giswqs/geemap/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.7.6 Get Started!

Ready to contribute? Here's how to set up *geemap* for local development.

1. Fork the *geemap* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/geemap.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv geemap
$ cd geemap/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 geemap tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.7.7 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/giswqs/geemap/pull_requests and make sure that the tests pass for all supported Python versions.

1.7.8 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_geemap
```

1.7.9 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

1.8 References

To support my work, please consider citing the following articles:

- **Wu, Q.**, (2020). geemap: A Python package for interactive mapping with Google Earth Engine. *The Journal of Open Source Software*, 5(51), 2305. https://doi.org/10.21105/joss.02305
- **Wu, Q.**, Lane, C. R., Li, X., Zhao, K., Zhou, Y., Clinton, N., DeVries, B., Golden, H. E., & Lang, M. W. (2019). Integrating LiDAR data and multi-temporal aerial imagery to map wetland inundation dynamics using Google Earth Engine. *Remote Sensing of Environment*, 228, 1-13. https://doi.org/10.1016/j.rse.2019.04.015 (pdf | source code)

1.9 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install geemap, run this command in your terminal:

```
$ pip install geemap
```

This is the preferred method to install geemap, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for geemap can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/giswqs/geemap
```

Or download the tarball:

```
$ curl -OJL https://github.com/giswqs/geemap/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use geemap in a project:

```
import geemap
```

CHAPTER 4

Modules

4.1 geemap package

4.1.1 Submodules

4.1.2 geemap.basemaps module

Module for basemaps. Each basemap is defined as item in the basemaps dictionary. For example, to access Google basemaps, use the following:

basemaps['ROADMAP'], basemaps['SATELLITE'], basemaps['HYBRID'].

More WMS basemaps can be found at the following websites:

1. USGS National Map: https://viewer.nationalmap.gov/services/
2. MRLC NLCD Land Cover data: https://viewer.nationalmap.gov/services/
3. FWS NWI Wetlands data: https://www.fws.gov/wetlands/Data/Web-Map-Services.html

geemap.basemaps.**get_qms**(*service_id*)

geemap.basemaps.**get_xyz_dict**(*free_only=True*, *_collection=None*, *_output=None*)
Returns a dictionary of xyz services.

Parameters **free_only** (*bool, optional*) – Whether to return only free xyz tile services that do not require an access token. Defaults to True.

Returns A dictionary of xyz services.

Return type dict

geemap.basemaps.**qms_to_geemap**(*service_id*)

geemap.basemaps.**search_qms**(*keywords*, *limit=10*)
Search qms files for keywords. Reference: https://github.com/geopandas/xyzservices/issues/65

Parameters

- **keywords** (*str*) – Keywords to search for.
- **limit** (*int*) – Number of results to return.

geemap.basemaps.**xyz_to_folium**()
Convert xyz tile services to folium tile layers.

Returns A dictionary of folium tile layers.

Return type dict

geemap.basemaps.**xyz_to_heremap**()
Convert xyz tile services to hermap tile layers.

Returns A dictionary of heremap tile layers.

Return type dict

geemap.basemaps.**xyz_to_leaflet**()
Convert xyz tile services to ipyleaflet tile layers.

Returns A dictionary of ipyleaflet tile layers.

Return type dict

geemap.basemaps.**xyz_to_plotly**()
Convert xyz tile services to plotly tile layers.

Returns A dictionary of plotly tile layers.

Return type dict

geemap.basemaps.**xyz_to_pydeck**()
Convert xyz tile services to pydeck custom tile layers.

Returns A dictionary of pydeck tile layers.

Return type dict

4.1.3 geemap.cli module

Console script for geemap.

4.1.4 geemap.conversion module

Module for converting Google Earth Engine (GEE) JavaScripts to Python scripts and Jupyter notebooks.

To convert a GEE JavaScript to Python script: js_to_python(in_file out_file)

To convert all GEE JavaScripts in a folder recursively to Python scripts: js_to_python_dir(in_dir, out_dir)

To convert a GEE Python script to Jupyter notebook: py_to_ipynb(in_file, template_file, out_file)

To convert all GEE Python scripts in a folder recursively to Jupyter notebooks: py_to_ipynb_dir(in_dir, template_file, out_dir)

To execute a Jupyter notebook and save output cells: execute_notebook(in_file)

To execute all Jupyter notebooks in a folder recursively: execute_notebook_dir(in_dir)

geemap.conversion.**check_map_functions**(*input_lines*)
Extracts Earth Engine map function

Parameters **input_lines** (*list*) – List of Earth Engine JavaScrips

Returns Output JavaScript with map function

Return type list

geemap.conversion.**convert_for_loop**(*line*)
Converts JavaScript for loop to Python for loop.

Parameters **line** (*str*) – Input JavaScript for loop

Returns Converted Python for loop.

Return type str

geemap.conversion.**create_new_cell**(*contents*, *replace=False*)
Create a new cell in Jupyter notebook based on the contents.

Parameters **contents** (*str*) – A string of Python code.

geemap.conversion.**download_gee_app**(*url*, *out_file=None*)
Downloads JavaScript source code from a GEE App

Parameters

- **url** (*str*) – The URL of the GEE App.
- **out_file** (*str, optional*) – The output file path for the downloaded JavaScript. Defaults to None.

geemap.conversion.**execute_notebook**(*in_file*)
Executes a Jupyter notebook and save output cells

Parameters **in_file** (*str*) – Input Jupyter notebook.

geemap.conversion.**execute_notebook_dir**(*in_dir*)
Executes all Jupyter notebooks in the given directory recursively and save output cells.

Parameters **in_dir** (*str*) – Input folder containing notebooks.

geemap.conversion.**find_matching_bracket**(*lines*, *start_line_index*, *start_char_index*, *matching_char='{'*)
Finds the position of the matching closing bracket from a list of lines.

Parameters

- **lines** (*list*) – The input list of lines.
- **start_line_index** (*int*) – The line index where the starting bracket is located.
- **start_char_index** (*int*) – The position index of the starting bracket.
- **matching_char** (*str, optional*) – The starting bracket to search for. Defaults to '{'.

Returns The line index where the matching closing bracket is located. matching_char_index (int): The position index of the matching closing bracket.

Return type matching_line_index (int)

geemap.conversion.**format_params**(*line*, *sep=':'*)
Formats keys in a dictionary and adds quotes to the keys. For example, {min: 0, max: 10} will result in ('min': 0, 'max': 10)

Parameters

- **line** (*str*) – A string.
- **sep** (*str, optional*) – Separator. Defaults to ':'.

Returns A string with keys quoted

Return type [str]

geemap.conversion.**get_js_examples**(*out_dir=None*)
Gets Earth Engine JavaScript examples from the geemap package.

Parameters **out_dir** (*str, optional*) – The folder to copy the JavaScript examples to. Defaults to None.

Returns The folder containing the JavaScript examples.

Return type str

geemap.conversion.**get_nb_template**(*download_latest=False*, *out_file=None*)
Get the Earth Engine Jupyter notebook template.

Parameters

- **download_latest** (*bool, optional*) – If True, downloads the latest notebook template from GitHub. Defaults to False.
- **out_file** (*str, optional*) – Set the output file path of the notebook template. Defaults to None.

Returns The file path of the template.

Return type str

geemap.conversion.**js_snippet_to_py**(*in_js_snippet*, *add_new_cell=True*, *import_ee=True*, *import_geemap=False*, *show_map=True*)
Converts an Earth Engine JavaScript snippet wrapped in triple quotes to Python directly on a Jupyter notebook.

Parameters

- **in_js_snippet** (*str*) – Earth Engine JavaScript within triple quotes.
- **add_new_cell** (*bool, optional*) – Whether add the converted Python to a new cell.
- **import_ee** (*bool, optional*) – Whether to import ee. Defaults to True.
- **import_geemap** (*bool, optional*) – Whether to import geemap. Defaults to False.
- **show_map** (*bool, optional*) – Whether to show the map. Defaults to True.

Returns A list of Python script.

Return type list

geemap.conversion.**js_to_python**(*in_file*, *out_file=None*, *use_qgis=True*, *github_repo=None*, *show_map=True*)
Converts an Earth Engine JavaScript to Python script.

Args: in_file (str): File path of the input JavaScript. out_file (str, optional): File path of the output Python script. Defaults to None. use_qgis (bool, optional): Whether to add "from ee_plugin import Map

" to the output script. Defaults to True.

github_repo (str, optional): GitHub repo url. Defaults to None. show_map (bool, optional): Whether to add "Map" to the output script. Defaults to True.

Returns: list: Python script

geemap.conversion.**js_to_python_dir**(*in_dir*, *out_dir=None*, *use_qgis=True*, *github_repo=None*)

Converts all Earth Engine JavaScripts in a folder recursively to Python scripts.

Args: in_dir (str): The input folder containing Earth Engine JavaScripts. out_dir (str, optional): The output folder containing Earth Engine Python scripts. Defaults to None. use_qgis (bool, optional): Whether to add "from ee_plugin import Map

" to the output script. Defaults to True. github_repo (str, optional): GitHub repo url. Defaults to None.

geemap.conversion.**py_to_ipynb**(*in_file*, *template_file=None*, *out_file=None*, *github_username=None*, *github_repo=None*)

Converts Earth Engine Python script to Jupyter notebook.

Parameters

- **in_file** (*str*) – Input Earth Engine Python script.
- **template_file** (*str*) – Input Jupyter notebook template.
- **out_file** (*str, optional)*) – Output Jupyter notebook.
- **github_username** (*str, optional*) – GitHub username. Defaults to None.
- **github_repo** (*str, optional*) – GitHub repo name. Defaults to None.

geemap.conversion.**py_to_ipynb_dir**(*in_dir*, *template_file=None*, *out_dir=None*, *github_username=None*, *github_repo=None*)

Converts Earth Engine Python scripts in a folder recursively to Jupyter notebooks.

Parameters

- **in_dir** (*str*) – Input folder containing Earth Engine Python scripts.
- **str, optional)** (*out_dir*) – Output folder. Defaults to None.
- **template_file** (*str*) – Input jupyter notebook template file.
- **github_username** (*str, optional*) – GitHub username. Defaults to None.
- **github_repo** (*str, optional*) – GitHub repo name. Defaults to None.

geemap.conversion.**remove_qgis_import**(*in_file*)

Removes 'from ee_plugin import Map' from an Earth Engine Python script.

Parameters **in_file** (*str*) – Input file path of the Python script.

Returns List of lines 'from ee_plugin import Map' removed.

Return type list

geemap.conversion.**template_footer**(*in_template*)

Extracts footer from the notebook template.

Parameters **in_template** (*str*) – Input notebook template file path.

Returns List of lines.

Return type list

geemap.conversion.**template_header**(*in_template*)

Extracts header from the notebook template.

Parameters **in_template** (*str*) – Input notebook template file path.

Returns List of lines.

Return type list

geemap.conversion.**update_nb_header**(*in_file*, *github_username=None*, *github_repo=None*)
Updates notebook header (binder and Google Colab URLs).

Parameters

- **in_file** (*str*) – The input Jupyter notebook.
- **github_username** (*str, optional*) – GitHub username. Defaults to None.
- **github_repo** (*str, optional*) – GitHub repo name. Defaults to None.

geemap.conversion.**update_nb_header_dir**(*in_dir*, *github_username=None*, *github_repo=None*)
Updates header (binder and Google Colab URLs) of all notebooks in a folder .

Parameters

- **in_dir** (*str*) – The input directory containing Jupyter notebooks.
- **github_username** (*str, optional*) – GitHub username. Defaults to None.
- **github_repo** (*str, optional*) – GitHub repo name. Defaults to None.

geemap.conversion.**use_math**(*lines*)
Checks if an Earth Engine uses Math library

Parameters **lines** (*list*) – An Earth Engine JavaScript.

Returns Returns True if the script contains 'Math.'. For example 'Math.PI', 'Math.pow'

Return type [bool]

4.1.5 geemap.foliumap module

This module extends the folium Map class. It is designed to be used in Google Colab, as Google Colab currently does not support ipyleaflet.

class geemap.foliumap.**CustomControl**(*html*, *position='bottomleft'*)
Bases: branca.element.MacroElement

Put any HTML on the map as a Leaflet Control. Adopted from https://github.com/python-visualization/folium/pull/1662

class geemap.foliumap.**FloatText**(*text*, *bottom=75*, *left=75*)
Bases: branca.element.MacroElement

Adds a floating image in HTML canvas on top of the map.

class geemap.foliumap.**Map**(***kwargs*)
Bases: folium.folium.Map

The Map class inherits from folium.Map. By default, the Map will add Google Maps as the basemap. Set add_google_map = False to use OpenStreetMap as the basemap.

Returns folium map object.

Return type object

addLayer(*ee_object*, *vis_params={}*, *name='Layer untitled'*, *shown=True*, *opacity=1.0*, ***kwargs*)
Adds a given EE object to the map as a layer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer untitled'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

addLayerControl()
Adds layer control to the map.

add_basemap(*basemap='HYBRID'*)
Adds a basemap to the map.

Parameters **basemap** (*str, optional*) – Can be one of string from ee_basemaps. Defaults to 'HYBRID'.

add_census_data(*wms*, *layer*, *census_dict=None*, ***kwargs*)
Adds a census data layer to the map.

Parameters

- **wms** (*str*) – The wms to use. For example, "Current", "ACS 2021", "Census 2020". See the complete list at https://tigerweb.geo.census.gov/tigerwebmain/TIGERweb_wms.html
- **layer** (*str*) – The layer name to add to the map.
- **census_dict** (*dict, optional*) – A dictionary containing census data. Defaults to None. It can be obtained from the get_census_dict() function.

add_circle_markers_from_xy(*data*, *x='longitude'*, *y='latitude'*, *radius=10*, *popup=None*, *tooltip=None*, *min_width=100*, *max_width=200*, ***kwargs*)
Adds a marker cluster to the map.

Parameters

- **data** (*str | pd.DataFrame*) – A csv or Pandas DataFrame containing x, y, z values.
- **x** (*str, optional*) – The column name for the x values. Defaults to "longitude".
- **y** (*str, optional*) – The column name for the y values. Defaults to "latitude".
- **radius** (*int, optional*) – The radius of the circle. Defaults to 10.
- **popup** (*list, optional*) – A list of column names to be used as the popup. Defaults to None.
- **tooltip** (*list, optional*) – A list of column names to be used as the tooltip. Defaults to None.
- **min_width** (*int, optional*) – The minimum width of the popup. Defaults to 100.
- **max_width** (*int, optional*) – The maximum width of the popup. Defaults to 200.

add_cog_layer(*url*, *name='Untitled'*, *attribution='.'*, *opacity=1.0*, *shown=True*, *bands=None*, *titiler_endpoint=None*, ***kwargs*)
Adds a COG TileLayer to the map.

Parameters

- **url** (*str*) – The URL of the COG tile layer.

- **name** (*str, optional*) – The layer name to use for the layer. Defaults to 'Untitled'.
- **attribution** (*str, optional*) – The attribution to use. Defaults to '.'.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **bands** (*list, optional*) – A list of bands to use. Defaults to None.
- **titiler_endpoint** (*str, optional*) – Titiler endpoint. Defaults to "https://titiler.xyz".

add_cog_mosaic(***kwargs*)

add_colorbar(*vis_params*, *index=None*, *label=''*, *categorical=False*, *step=None*, *background_color=None*, ***kwargs*)

Add a colorbar to the map.

Parameters

- **colors** (*list*) – The set of colors to be used for interpolation. Colors can be provided in the form: * tuples of RGBA ints between 0 and 255 (e.g: (255, 255, 0) or (255, 255, 0, 255)) * tuples of RGBA floats between 0. and 1. (e.g: (1.,1.,0.) or (1., 1., 0., 1.)) * HTML-like string (e.g: "#ffff00) * a color name or shortcut (e.g: "y" or "yellow")
- **vmin** (*int, optional*) – The minimal value for the colormap. Values lower than vmin will be bound directly to colors[0].. Defaults to 0.
- **vmax** (*float, optional*) – The maximal value for the colormap. Values higher than vmax will be bound directly to colors[-1]. Defaults to 1.0.
- **index** (*list, optional*) – The values corresponding to each color. It has to be sorted, and have the same length as colors. If None, a regular grid between vmin and vmax is created.. Defaults to None.
- **label** (*str, optional*) – The caption for the colormap. Defaults to "".
- **categorical** (*bool, optional*) – Whether or not to create a categorical colormap. Defaults to False.
- **step** (*int, optional*) – The step to split the LinearColormap into a StepColormap. Defaults to None.

add_colorbar_branca(*vis_params*, *index=None*, *label=''*, *categorical=False*, *step=None*, *background_color=None*, ***kwargs*)

Add a colorbar to the map.

Parameters

- **colors** (*list*) – The set of colors to be used for interpolation. Colors can be provided in the form: * tuples of RGBA ints between 0 and 255 (e.g: (255, 255, 0) or (255, 255, 0, 255)) * tuples of RGBA floats between 0. and 1. (e.g: (1.,1.,0.) or (1., 1., 0., 1.)) * HTML-like string (e.g: "#ffff00) * a color name or shortcut (e.g: "y" or "yellow")
- **vmin** (*int, optional*) – The minimal value for the colormap. Values lower than vmin will be bound directly to colors[0].. Defaults to 0.
- **vmax** (*float, optional*) – The maximal value for the colormap. Values higher than vmax will be bound directly to colors[-1]. Defaults to 1.0.
- **index** (*list, optional*) – The values corresponding to each color. It has to be sorted, and have the same length as colors. If None, a regular grid between vmin and vmax is created.. Defaults to None.

- **label** (*str, optional*) – The caption for the colormap. Defaults to "".
- **categorical** (*bool, optional*) – Whether or not to create a categorical colormap. Defaults to False.
- **step** (*int, optional*) – The step to split the LinearColormap into a StepColormap. Defaults to None.

add_colormap(*width=4.0*, *height=0.3*, *vmin=0*, *vmax=1.0*, *palette=None*, *vis_params=None*, *cmap='gray'*, *discrete=False*, *label=None*, *label_size=10*, *label_weight='normal'*, *tick_size=8*, *bg_color='white'*, *orientation='horizontal'*, *dpi='figure'*, *transparent=False*, *position=(70, 5)*, ***kwargs*)

Add a colorbar to the map. Under the hood, it uses matplotlib to generate the colorbar, save it as a png file, and add it to the map using m.add_image().

Parameters

- **width** (*float*) – Width of the colorbar in inches. Default is 4.0.
- **height** (*float*) – Height of the colorbar in inches. Default is 0.3.
- **vmin** (*float*) – Minimum value of the colorbar. Default is 0.
- **vmax** (*float*) – Maximum value of the colorbar. Default is 1.0.
- **palette** (*list*) – List of colors to use for the colorbar. It can also be a cmap name, such as ndvi, ndwi, dem, coolwarm. Default is None.
- **vis_params** (*dict*) – Visualization parameters as a dictionary. See https://developers.google.com/earth-engine/guides/image_visualization for options.
- **cmap** (*str, optional*) – Matplotlib colormap. Defaults to "gray". See https://matplotlib.org/3.3.4/tutorials/colors/colormaps.html#sphx-glr-tutorials-colors-colormaps-py for options.
- **discrete** (*bool, optional*) – Whether to create a discrete colorbar. Defaults to False.
- **label** (*str, optional*) – Label for the colorbar. Defaults to None.
- **label_size** (*int, optional*) – Font size for the colorbar label. Defaults to 12.
- **label_weight** (*str, optional*) – Font weight for the colorbar label, can be "normal", "bold", etc. Defaults to "normal".
- **tick_size** (*int, optional*) – Font size for the colorbar tick labels. Defaults to 10.
- **bg_color** (*str, optional*) – Background color for the colorbar. Defaults to "white".
- **orientation** (*str, optional*) – Orientation of the colorbar, such as "vertical" and "horizontal". Defaults to "horizontal".
- **dpi** (*float | str, optional*) – The resolution in dots per inch. If 'figure', use the figure's dpi value. Defaults to "figure".
- **transparent** (*bool, optional*) – Whether to make the background transparent. Defaults to False.
- **position** (*tuple, optional*) – The position of the colormap in the format of (x, y), the percentage ranging from 0 to 100, starting from the lower-left corner. Defaults to (0, 0).
- ****kwargs** – Other keyword arguments to pass to matplotlib.pyplot.savefig().

Returns Path to the output image.

Return type str

add_data(*data*, *column*, *colors=None*, *labels=None*, *cmap=None*, *scheme='Quantiles'*, *k=5*, *add_legend=True*, *legend_title=None*, *legend_kwds=None*, *classification_kwds=None*, *style_function=None*, *highlight_function=None*, *layer_name='Untitled'*, *info_mode='on_hover'*, *encoding='utf-8'*, ***kwargs*)

Add vector data to the map with a variety of classification schemes.

Parameters

- **data** (*str | pd.DataFrame | gpd.GeoDataFrame*) – The data to classify. It can be a filepath to a vector dataset, a pandas dataframe, or a geopandas geodataframe.
- **column** (*str*) – The column to classify.
- **cmap** (*str, optional*) – The name of a colormap recognized by matplotlib. Defaults to None.
- **colors** (*list, optional*) – A list of colors to use for the classification. Defaults to None.
- **labels** (*list, optional*) – A list of labels to use for the legend. Defaults to None.
- **scheme** (*str, optional*) – Name of a choropleth classification scheme (requires mapclassify). Name of a choropleth classification scheme (requires mapclassify). A mapclassify.MapClassifier object will be used under the hood. Supported are all schemes provided by mapclassify (e.g. 'BoxPlot', 'EqualInterval', 'FisherJenks', 'FisherJenksSampled', 'HeadTailBreaks', 'JenksCaspall', 'JenksCaspallForced', 'JenksCaspallSampled', 'MaxP', 'MaximumBreaks', 'NaturalBreaks', 'Quantiles', 'Percentiles', 'StdMean', 'UserDefined'). Arguments can be passed in classification_kwds.
- **k** (*int, optional*) – Number of classes (ignored if scheme is None or if column is categorical). Default to 5.
- **legend_kwds** (*dict, optional*) – Keyword arguments to pass to `matplotlib.pyplot.legend()` or *matplotlib.pyplot.colorbar*. Defaults to None. Keyword arguments to pass to `matplotlib.pyplot.legend()` or Additional accepted keywords when *scheme* is specified: fmt : string

  A formatting specification for the bin edges of the classes in the legend. For example, to have no decimals: `{"fmt": "{:.0f}"}`.

  labels [list-like] A list of legend labels to override the auto-generated labblels. Needs to have the same number of elements as the number of classes (*k*).

  interval [boolean (default False)] An option to control brackets from mapclassify legend. If True, open/closed interval brackets are shown in the legend.

- **classification_kwds** (*dict, optional*) – Keyword arguments to pass to mapclassify. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style_function** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None. style_callback is a function that takes the feature as argument and should return a dictionary of the following form: style_callback = lambda feat: {"fillColor": feat["properties"]["color"]} style is a dictionary of the following form:

  style = { "stroke": False, "color": "#ff0000", "weight": 1, "opacity": 1, "fill": True, "fillColor": "#ffffff", "fillOpacity": 1.0, "dashArray": "9" "clickable": True,

}

- **hightlight_function** (*function, optional*) – Highlighting function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None. highlight_function is a function that takes the feature as argument and should return a dictionary of the following form: highlight_function = lambda feat: {"fillColor": feat["properties"]["color"]}
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding of the GeoJSON file. Defaults to "utf-8".

add_gdf(*gdf*, *layer_name='Untitled'*, *zoom_to_layer=True*, *info_mode='on_hover'*, *fields=None*, ***kwargs*)

Adds a GeoPandas GeoDataFrame to the map.

Parameters

- **gdf** (*GeoDataFrame*) – A GeoPandas GeoDataFrame.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Untitled".
- **zoom_to_layer** (*bool, optional*) – Whether to zoom to the layer.
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **fields** (*list, optional*) – The fields to be displayed in the popup. Defaults to None.

add_gdf_from_postgis(*sql*, *con*, *layer_name='Untitled'*, *zoom_to_layer=True*, ***kwargs*)

Adds a GeoPandas GeoDataFrameto the map.

Parameters

- **sql** (*str*) – SQL query to execute in selecting entries from database, or name of the table to read from the database.
- **con** (*sqlalchemy.engine.Engine*) – Active connection to the database to query.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Untitled".
- **zoom_to_layer** (*bool, optional*) – Whether to zoom to the layer.

add_geojson(*in_geojson*, *layer_name='Untitled'*, *encoding='utf-8'*, *info_mode='on_hover'*, *fields=None*, ***kwargs*)

Adds a GeoJSON file to the map.

Parameters

- **in_geojson** (*str*) – The input file path to the GeoJSON.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Untitled".
- **encoding** (*str, optional*) – The encoding of the GeoJSON file. Defaults to "utf-8".
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

- **fields** (*list, optional*) – The fields to be displayed in the popup. Defaults to None.

Raises FileNotFoundError – The provided GeoJSON file could not be found.

add_heatmap(*data*, *latitude='latitude'*, *longitude='longitude'*, *value='value'*, *name='Heat map'*, *radius=25*, ***kwargs*)
Adds a heat map to the map. Reference: https://stackoverflow.com/a/54756617

Parameters

- **data** (*str | list | pd.DataFrame*) – File path or HTTP URL to the input file or a list of data points in the format of [[x1, y1, z1], [x2, y2, z2]]. For example, https://raw.githubusercontent.com/giswqs/leafmap/master/examples/data/world_cities.csv
- **latitude** (*str, optional*) – The column name of latitude. Defaults to "latitude".
- **longitude** (*str, optional*) – The column name of longitude. Defaults to "longitude".
- **value** (*str, optional*) – The column name of values. Defaults to "value".
- **name** (*str, optional*) – Layer name to use. Defaults to "Heat map".
- **radius** (*int, optional*) – Radius of each "point" of the heatmap. Defaults to 25.

Raises ValueError – If data is not a list.

add_html(*html*, *position='bottomright'*, ***kwargs*)
Add HTML to the map.

Parameters

- **html** (*str*) – The HTML to add.
- **position** (*str, optional*) – The position of the widget. Defaults to "bottomright".

add_image(*image*, *position=(0, 0)*, ***kwargs*)
Add an image to the map.

Parameters

- **image** (*str | ipywidgets.Image*) – The image to add.
- **position** (*tuple, optional*) – The position of the image in the format of (x, y), the percentage ranging from 0 to 100, starting from the lower-left corner. Defaults to (0, 0).

add_kml(*in_kml*, *layer_name='Untitled'*, *info_mode='on_hover'*, *fields=None*, ***kwargs*)
Adds a KML file to the map.

Parameters

- **in_kml** (*str*) – The input file path to the KML.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Untitled".
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **fields** (*list, optional*) – The fields to be displayed in the popup. Defaults to None.

Raises FileNotFoundError – The provided KML file could not be found.

add_labels(*data*, *column*, *font_size='12pt'*, *font_color='black'*, *font_family='arial'*, *font_weight='normal'*, *x='longitude'*, *y='latitude'*, *draggable=True*, *layer_name='Labels'*, ***kwargs*)

Adds a label layer to the map. Reference: https://python-visualization.github.io/folium/modules.html#folium.features.DivIcon

Parameters

- **data** (*pd.DataFrame | ee.FeatureCollection*) – The input data to label.
- **column** (*str*) – The column name of the data to label.
- **font_size** (*str, optional*) – The font size of the labels. Defaults to "12pt".
- **font_color** (*str, optional*) – The font color of the labels. Defaults to "black".
- **font_family** (*str, optional*) – The font family of the labels. Defaults to "arial".
- **font_weight** (*str, optional*) – The font weight of the labels, can be normal, bold. Defaults to "normal".
- **x** (*str, optional*) – The column name of the longitude. Defaults to "longitude".
- **y** (*str, optional*) – The column name of the latitude. Defaults to "latitude".
- **draggable** (*bool, optional*) – Whether the labels are draggable. Defaults to True.
- **layer_name** (*str, optional*) – The name of the layer. Defaults to "Labels".

add_layer(*ee_object*, *vis_params={}*, *name='Layer untitled'*, *shown=True*, *opacity=1.0*, ***kwargs*)

Adds a given EE object to the map as a layer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer untitled'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

add_layer_control()

Adds layer control to the map.

add_legend(*title='Legend'*, *labels=None*, *colors=None*, *legend_dict=None*, *builtin_legend=None*, *opacity=1.0*, *position='bottomright'*, *draggable=True*, *style={}*)

Adds a customized legend to the map. Reference: https://bit.ly/3oV6vnH. If you want to add multiple legends to the map, you need to set the *draggable* argument to False.

Parameters

- **title** (*str, optional*) – Title of the legend. Defaults to 'Legend'. Defaults to "Legend".
- **colors** (*list, optional*) – A list of legend colors. Defaults to None.
- **labels** (*list, optional*) – A list of legend labels. Defaults to None.

- **legend_dict** (*dict, optional*) – A dictionary containing legend items as keys and color as values. If provided, legend_keys and legend_colors will be ignored. Defaults to None.
- **builtin_legend** (*str, optional*) – Name of the builtin legend to add to the map. Defaults to None.
- **opacity** (*float, optional*) – The opacity of the legend. Defaults to 1.0.
- **position** (*str, optional*) – The position of the legend, can be one of the following: "topleft", "topright", "bottomleft", "bottomright". Defaults to "bottomright".
- **draggable** (*bool, optional*) – If True, the legend can be dragged to a new position. Defaults to True.
- **style** – Additional keyword arguments to style the legend, such as position, bottom, right, z-index, border, background-color, border-radius, padding, font-size, etc. The default style is: style = {

  'position': 'fixed', 'z-index': '9999', 'border': '2px solid grey', 'background-color': 'rgba(255, 255, 255, 0.8)', 'border-radius': '5px', 'padding': '10px', 'font-size': '14px', 'bottom': '20px', 'right': '5px'

  }

add_marker(*location*, *popup=None*, *tooltip=None*, *icon=None*, *draggable=False*, ***kwargs*)

Adds a marker to the map. More info about marker options at https://python-visualization.github.io/folium/modules.html#folium.map.Marker.

Parameters

- **location** (*list | tuple*) – The location of the marker in the format of [lat, lng].
- **popup** (*str, optional*) – The popup text. Defaults to None.
- **tooltip** (*str, optional*) – The tooltip text. Defaults to None.
- **icon** (*str, optional*) – The icon to use. Defaults to None.
- **draggable** (*bool, optional*) – Whether the marker is draggable. Defaults to False.

add_markers_from_xy(*data, x='longitude', y='latitude', popup=None, min_width=100, max_width=200, layer_name='Marker Cluster', color_column=None, marker_colors=None, icon_colors=['white'], icon_names=['info'], angle=0, prefix='fa', add_legend=True, **kwargs*)

Adds a marker cluster to the map.

Parameters

- **data** (*str | pd.DataFrame*) – A csv or Pandas DataFrame containing x, y, z values.
- **x** (*str, optional*) – The column name for the x values. Defaults to "longitude".
- **y** (*str, optional*) – The column name for the y values. Defaults to "latitude".
- **popup** (*list, optional*) – A list of column names to be used as the popup. Defaults to None.
- **min_width** (*int, optional*) – The minimum width of the popup. Defaults to 100.
- **max_width** (*int, optional*) – The maximum width of the popup. Defaults to 200.
- **layer_name** (*str, optional*) – The name of the layer. Defaults to "Marker Cluster".

- **color_column** (*str, optional*) – The column name for the color values. Defaults to None.
- **marker_colors** (*list, optional*) – A list of colors to be used for the markers. Defaults to None.
- **icon_colors** (*list, optional*) – A list of colors to be used for the icons. Defaults to ['white'].
- **icon_names** (*list, optional*) – A list of names to be used for the icons. More icons can be found at https://fontawesome.com/v4/icons or https://getbootstrap.com/docs/3.3/components/?utm_source=pocket_mylist. Defaults to ['info'].
- **angle** (*int, optional*) – The angle of the icon. Defaults to 0.
- **prefix** (*str, optional*) – The prefix states the source of the icon. 'fa' for font-awesome or 'glyphicon' for bootstrap 3. Defaults to 'fa'.
- **add_legend** (*bool, optional*) – If True, a legend will be added to the map. Defaults to True.

add_netcdf (*filename*, *variables=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name='NetCDF layer'*, *shift_lon=True*, *lat='lat'*, *lon='lon'*, ***kwargs*)

Generate an ipyleaflet/folium TileLayer from a netCDF file. If you are using this function in JupyterHub on a remote server (e.g., Binder, Microsoft Planetary Computer), try adding to following two lines to the beginning of the notebook if the raster does not render properly.

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = f'{os.environ['JUPYTERHUB_SERVICE_PREFIX'].lstrip('/')}/proxy/{{port}}'

Parameters

- **filename** (*str*) – File path or HTTP URL to the netCDF file.
- **variables** (*int, optional*) – The variable/band names to extract data from the netCDF file. Defaults to None. If None, all variables will be extracted.
- **port** (*str, optional*) – The port to use for the server. Defaults to "default".
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to "netCDF layer".
- **shift_lon** (*bool, optional*) – Flag to shift longitude values from [0, 360] to the range [-180, 180]. Defaults to True.
- **lat** (*str, optional*) – Name of the latitude variable. Defaults to 'lat'.
- **lon** (*str, optional*) – Name of the longitude variable. Defaults to 'lon'.

add_osm(*query*, *layer_name='Untitled'*, *which_result=None*, *by_osmid=False*, *buffer_dist=None*, *to_ee=False*, *geodesic=True*, ***kwargs*)

Adds OSM data to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **which_result** (*INT, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **by_osmid** (*bool, optional*) – If True, handle query as an OSM ID for lookup rather than text search. Defaults to False.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.
- **to_ee** (*bool, optional*) – Whether to convert the csv to an ee.FeatureCollection.
- **geodesic** (*bool, optional*) – Whether line segments should be interpreted as spherical geodesics. If false, indicates that line segments should be interpreted as planar lines in the specified CRS. If absent, defaults to true if the CRS is geographic (including the default EPSG:4326), or to false if the CRS is projected.

add_osm_from_address(*address, tags, dist=1000, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within some distance N, S, E, W of address to the map.

Parameters

- **address** (*str*) – The address to geocode and use as the central point around which to get the geometries.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **dist** (*int, optional*) – Distance in meters. Defaults to 1000.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].

- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_bbox(*north, south, east, west, tags, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within a N, S, E, W bounding box to the map.

Parameters

- **north** (*float*) – Northern latitude of bounding box.
- **south** (*float*) – Southern latitude of bounding box.
- **east** (*float*) – Eastern longitude of bounding box.
- **west** (*float*) – Western longitude of bounding box.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_geocode(*query, which_result=None, by_osmid=False, buffer_dist=None, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM data of place(s) by name or ID to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **which_result** (*int, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **by_osmid** (*bool, optional*) – If True, handle query as an OSM ID for lookup rather than text search. Defaults to False.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.

- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_place(*query, tags, which_result=None, buffer_dist=None, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within boundaries of geocodable place(s) to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **which_result** (*int, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_point(*center_point, tags, dist=1000, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within some distance N, S, E, W of a point to the map.

Parameters

- **center_point** (*tuple*) – The (lat, lng) center point around which to get the geometries.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **dist** (*int, optional*) – Distance in meters. Defaults to 1000.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_polygon(*polygon, tags, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within boundaries of a (multi)polygon to the map.

Parameters

- **polygon** (*shapely.geometry.Polygon | shapely.geometry.MultiPolygon*) – Geographic boundaries to fetch geometries within
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.

- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_planet_by_month(*year=2016, month=1, name=None, api_key=None, token_name='PLANET_API_KEY'*)

Adds a Planet global mosaic by month to the map. To get a Planet API key, see https://developers.planet.com/quickstart/apis

Parameters

- **year** (*int, optional*) – The year of Planet global mosaic, must be >=2016. Defaults to 2016.
- **month** (*int, optional*) – The month of Planet global mosaic, must be 1-12. Defaults to 1.
- **name** (*str, optional*) – The layer name to use. Defaults to None.
- **api_key** (*str, optional*) – The Planet API key. Defaults to None.
- **token_name** (*str, optional*) – The environment variable name of the API key. Defaults to "PLANET_API_KEY".

add_planet_by_quarter(*year=2016, quarter=1, name=None, api_key=None, token_name='PLANET_API_KEY'*)

Adds a Planet global mosaic by quarter to the map. To get a Planet API key, see https://developers.planet.com/quickstart/apis

Parameters

- **year** (*int, optional*) – The year of Planet global mosaic, must be >=2016. Defaults to 2016.
- **quarter** (*int, optional*) – The quarter of Planet global mosaic, must be 1-12. Defaults to 1.
- **name** (*str, optional*) – The layer name to use. Defaults to None.
- **api_key** (*str, optional*) – The Planet API key. Defaults to None.
- **token_name** (*str, optional*) – The environment variable name of the API key. Defaults to "PLANET_API_KEY".

add_points_from_xy(*data, x='longitude', y='latitude', popup=None, min_width=100, max_width=200, layer_name='Marker Cluster', color_column=None, marker_colors=None, icon_colors=['white'], icon_names=['info'], angle=0, prefix='fa', add_legend=True, **kwargs*)

Adds a marker cluster to the map.

Parameters

- **data** (*str | pd.DataFrame*) – A csv or Pandas DataFrame containing x, y, z values.
- **x** (*str, optional*) – The column name for the x values. Defaults to "longitude".
- **y** (*str, optional*) – The column name for the y values. Defaults to "latitude".

- **popup** (*list, optional*) – A list of column names to be used as the popup. Defaults to None.
- **min_width** (*int, optional*) – The minimum width of the popup. Defaults to 100.
- **max_width** (*int, optional*) – The maximum width of the popup. Defaults to 200.
- **layer_name** (*str, optional*) – The name of the layer. Defaults to "Marker Cluster".
- **color_column** (*str, optional*) – The column name for the color values. Defaults to None.
- **marker_colors** (*list, optional*) – A list of colors to be used for the markers. Defaults to None.
- **icon_colors** (*list, optional*) – A list of colors to be used for the icons. Defaults to ['white'].
- **icon_names** (*list, optional*) – A list of names to be used for the icons. More icons can be found at https://fontawesome.com/v4/icons or https://getbootstrap.com/docs/3.3/components/?utm_source=pocket_mylist. Defaults to ['info'].
- **angle** (*int, optional*) – The angle of the icon. Defaults to 0.
- **prefix** (*str, optional*) – The prefix states the source of the icon. 'fa' for font-awesome or 'glyphicon' for bootstrap 3. Defaults to 'fa'.
- **add_legend** (*bool, optional*) – If True, a legend will be added to the map. Defaults to True.

add_raster(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name='Local COG'*, ***kwargs*)

Add a local raster dataset to the map.

If you are using this function in JupyterHub on a remote server and the raster does not render properly, try running the following two lines before calling this function:

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = 'proxy/{port}'

Parameters

- **source** (*str*) – The path to the GeoTIFF file or the URL of the Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to 'Local COG'.

add_remote_tile(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name=None*, ***kwargs*)

Add a remote Cloud Optimized GeoTIFF (COG) to the map.

Parameters

- **source** (*str*) – The path to the remote Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to None.

add_shapefile(*in_shp*, *layer_name='Untitled'*, ***kwargs*)

Adds a shapefile to the map. See https://python-visualization.github.io/folium/modules.html#folium.features.GeoJson for more info about setting style.

Parameters

- **in_shp** (*str*) – The input file path to the shapefile.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Untitled".

Raises `FileNotFoundError` – The provided shapefile could not be found.

add_stac_layer(*url=None*, *collection=None*, *item=None*, *assets=None*, *bands=None*, *titiler_endpoint=None*, *name='STAC Layer'*, *attribution='.'*, *opacity=1.0*, *shown=True*, ***kwargs*)

Adds a STAC TileLayer to the map.

Parameters

- **url** (*str*) – HTTP URL to a STAC item, e.g., https://canada-spot-ortho.s3.amazonaws.com/canada_spot_orthoimages/canada_spot5_orthoimages/S5_2007/S5_11055_6057_20070622/S5_11055_6057_20070622.json
- **collection** (*str*) – The Microsoft Planetary Computer STAC collection ID, e.g., landsat-8-c2-l2.
- **item** (*str*) – The Microsoft Planetary Computer STAC item ID, e.g., LC08_L2SP_047027_20201204_02_T1.
- **assets** (*str | list*) – The Microsoft Planetary Computer STAC asset ID, e.g., ["SR_B7", "SR_B5", "SR_B4"].
- **bands** (*list*) – A list of band names, e.g., ["SR_B7", "SR_B5", "SR_B4"]
- **titiler_endpoint** (*str, optional*) – Titiler endpoint, e.g., "https://titiler.xyz", "planetary-computer", "pc". Defaults to None.

- **name** (*str, optional*) – The layer name to use for the layer. Defaults to 'STAC Layer'.
- **attribution** (*str, optional*) – The attribution to use. Defaults to ''.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.

add_styled_vector(*ee_object*, *column*, *palette*, *layer_name='Untitled'*, ***kwargs*)
Adds a styled vector to the map.

Parameters

- **ee_object** (*object*) – An ee.FeatureCollection.
- **column** (*str*) – The column name to use for styling.
- **palette** (*list | dict*) – The palette (e.g., list of colors or a dict containing label and color pairs) to use for styling.
- **layer_name** (*str, optional*) – The name to be used for the new layer. Defaults to "Untitled".

add_text(*text*, *fontsize=20*, *fontcolor='black'*, *bold=False*, *padding='5px'*, *background=True*, *bg_color='white'*, *border_radius='5px'*, *position='bottomright'*, ***kwargs*)
Add text to the map.

Parameters

- **text** (*str*) – The text to add.
- **fontsize** (*int, optional*) – The font size. Defaults to 20.
- **fontcolor** (*str, optional*) – The font color. Defaults to "black".
- **bold** (*bool, optional*) – Whether to use bold font. Defaults to False.
- **padding** (*str, optional*) – The padding. Defaults to "5px".
- **background** (*bool, optional*) – Whether to use background. Defaults to True.
- **bg_color** (*str, optional*) – The background color. Defaults to "white".
- **border_radius** (*str, optional*) – The border radius. Defaults to "5px".
- **position** (*str, optional*) – The position of the widget. Defaults to "bottomright".

add_tile_layer(*tiles='OpenStreetMap'*, *name='Untitled'*, *attribution='.'*, *overlay=True*, *control=True*, *shown=True*, *opacity=1.0*, *API_key=None*, ***kwargs*)
Add a XYZ tile layer to the map.

Parameters

- **tiles** (*str*) – The URL of the XYZ tile service.
- **name** (*str, optional*) – The layer name to use on the layer control. Defaults to 'Untitled'.
- **attribution** (*str, optional*) – The attribution of the data layer. Defaults to '.'.
- **overlay** (*str, optional*) – Allows overlay. Defaults to True.
- **control** (*str, optional*) – Adds the layer to the layer control. Defaults to True.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.

- **opacity** (*float, optional*) – Sets the opacity for the layer.
- **API_key** (*str, optional*) – – API key for Cloudmade or Mapbox tiles. Defaults to True.

add_time_slider(*ee_object*, *vis_params={}*, *region=None*, *layer_name='Time series'*, *labels=None*, *time_interval=1*, *position='bottomright'*, *slider_length='150px'*, *date_format='YYYY-MM-dd'*, *opacity=1.0*, ***kwargs*)

add_widget(*content*, *position='bottomright'*, ***kwargs*)

Add a widget (e.g., text, HTML, figure) to the map.

Parameters

- **content** (*str*) – The widget to add.
- **position** (*str, optional*) – The position of the widget. Defaults to "bottomright".

add_wms_layer(*url*, *layers*, *name=None*, *attribution=''*, *overlay=True*, *control=True*, *shown=True*, *format='image/png'*, *transparent=True*, *version='1.1.1'*, *styles=''*, ***kwargs*)

Add a WMS layer to the map.

Parameters

- **url** (*str*) – The URL of the WMS web service.
- **layers** (*str*) – Comma-separated list of WMS layers to show.
- **name** (*str, optional*) – The layer name to use on the layer control. Defaults to None.
- **attribution** (*str, optional*) – The attribution of the data layer. Defaults to ''.
- **overlay** (*str, optional*) – Allows overlay. Defaults to True.
- **control** (*str, optional*) – Adds the layer to the layer control. Defaults to True.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **format** (*str, optional*) – WMS image format (use 'image/png' for layers with transparency). Defaults to 'image/png'.
- **transparent** (*bool, optional*) – Whether the layer shall allow transparency. Defaults to True.
- **version** (*str, optional*) – Version of the WMS service to use. Defaults to "1.1.1".
- **styles** (*str, optional*) – Comma-separated list of WMS styles. Defaults to "".

add_xyz_service(*provider*, ***kwargs*)

Add a XYZ tile layer to the map.

Parameters **provider** (*str*) – A tile layer name starts with xyz or qms. For example, xyz.OpenTopoMap,

Raises `ValueError` – The provider is not valid. It must start with xyz or qms.

basemap_demo()

A demo for using geemap basemaps.

centerObject(*ee_object*, *zoom=None*)

Centers the map view on a given object.

Parameters

- **ee_object** (*Element|Geometry*) – An Earth Engine object to center on a geometry, image or feature.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

center_object(*ee_object*, *zoom=None*)
Centers the map view on a given object.

Parameters

- **ee_object** (*Element|Geometry*) – An Earth Engine object to center on a geometry, image or feature.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

dd_local_tile(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name='Local COG'*, ***kwargs*)
Add a local raster dataset to the map.

If you are using this function in JupyterHub on a remote server and the raster does not render properly, try running the following two lines before calling this function:

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = 'proxy/{port}'

Parameters

- **source** (*str*) – The path to the GeoTIFF file or the URL of the Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to 'Local COG'.

extract_values_to_points(*filename*)

publish(*name='Folium Map'*, *description=''*, *source_url=''*, *tags=None*, *source_file=None*, *open=True*, *formatting=None*, *token=None*, ***kwargs*)
Publish the map to datapane.com

Parameters

- **name** (*str, optional*) – The document name - can include spaces, caps, symbols, etc., e.g. "Profit & Loss 2020". Defaults to "Folium Map".
- **description** (*str, optional*) – A high-level description for the document, this is displayed in searches and thumbnails. Defaults to ''.
- **source_url** (*str, optional*) – A URL pointing to the source code for the document, e.g. a GitHub repo or a Colab notebook. Defaults to ''.

- **tags** (*bool, optional*) – A list of tags (as strings) used to categorise your document. Defaults to None.
- **source_file** (*str, optional*) – Path of jupyter notebook file to upload. Defaults to None.
- **open** (*bool, optional*) – Whether to open the map. Defaults to True.
- **formatting** (*ReportFormatting, optional*) – Set the basic styling for your report.
- **token** (*str, optional*) – The token to use to datapane to publish the map. See https://docs.datapane.com/tut-getting-started. Defaults to None.

remove_labels (***kwargs*)
Removes a layer from the map.

setCenter (*lon*, *lat*, *zoom=10*)
Centers the map view at a given coordinates with the given zoom level.

Parameters

- **lon** (*float*) – The longitude of the center, in degrees.
- **lat** (*float*) – The latitude of the center, in degrees.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to 10.

setControlVisibility (*layerControl=True*, *fullscreenControl=True*, *latLngPopup=True*)
Sets the visibility of the controls on the map.

Parameters

- **layerControl** (*bool, optional*) – Whether to show the control that allows the user to toggle layers on/off. Defaults to True.
- **fullscreenControl** (*bool, optional*) – Whether to show the control that allows the user to make the map full-screen. Defaults to True.
- **latLngPopup** (*bool, optional*) – Whether to show the control that pops up the Lat/lon when the user clicks on the map. Defaults to True.

setOptions (*mapTypeId='HYBRID'*, *styles={}*, *types=[]*)
Adds Google basemap to the map.

Parameters

- **mapTypeId** (*str, optional*) – A mapTypeId to set the basemap to. Can be one of "ROADMAP", "SATELLITE", "HYBRID" or "TERRAIN" to select one of the standard Google Maps API map types. Defaults to 'HYBRID'.
- **styles** (*[type], optional*) – A dictionary of custom MapTypeStyle objects keyed with a name that will appear in the map's Map Type Controls. Defaults to None.
- **types** (*[type], optional*) – A list of mapTypeIds to make available. If omitted, but opt_styles is specified, appends all of the style keys to the standard Google Maps API map types.. Defaults to None.

set_center (*lon*, *lat*, *zoom=10*)
Centers the map view at a given coordinates with the given zoom level.

Parameters

- **lon** (*float*) – The longitude of the center, in degrees.
- **lat** (*float*) – The latitude of the center, in degrees.

- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to 10.

set_control_visibility(*layerControl=True*, *fullscreenControl=True*, *latLngPopup=True*)
Sets the visibility of the controls on the map.

Parameters

- **layerControl** (*bool, optional*) – Whether to show the control that allows the user to toggle layers on/off. Defaults to True.
- **fullscreenControl** (*bool, optional*) – Whether to show the control that allows the user to make the map full-screen. Defaults to True.
- **latLngPopup** (*bool, optional*) – Whether to show the control that pops up the Lat/lon when the user clicks on the map. Defaults to True.

set_options(*mapTypeId='HYBRID'*, *styles={}*, *types=[]*)
Adds Google basemap to the map.

Parameters

- **mapTypeId** (*str, optional*) – A mapTypeId to set the basemap to. Can be one of "ROADMAP", "SATELLITE", "HYBRID" or "TERRAIN" to select one of the standard Google Maps API map types. Defaults to 'HYBRID'.
- **styles** (*[type], optional*) – A dictionary of custom MapTypeStyle objects keyed with a name that will appear in the map's Map Type Controls. Defaults to None.
- **types** (*[type], optional*) – A list of mapTypeIds to make available. If omitted, but opt_styles is specified, appends all of the style keys to the standard Google Maps API map types.. Defaults to None.

set_plot_options(***kwargs*)
Sets plotting options.

split_map(*left_layer='TERRAIN'*, *right_layer='OpenTopoMap'*, *left_args={}*, *right_args={}*, *left_label=None*, *right_label=None*, *left_position='bottomleft'*, *right_position='bottomright'*, ***kwargs*)
Adds a split-panel map.

Parameters

- **left_layer** (*str, optional*) – The left tile layer. Can be a local file path, HTTP URL, or a basemap name. Defaults to 'TERRAIN'.
- **right_layer** (*str, optional*) – The right tile layer. Can be a local file path, HTTP URL, or a basemap name. Defaults to 'OpenTopoMap'.
- **left_args** (*dict, optional*) – The arguments for the left tile layer. Defaults to {}.
- **right_args** (*dict, optional*) – The arguments for the right tile layer. Defaults to {}.

st_draw_features(*st_component*)
Get the draw features of the map.

Parameters **st_folium** (*streamlit-folium*) – The streamlit component.

Returns The draw features of the map.

Return type list

st_fit_bounds()
Fit the map to the bounds of the map.

Returns The map.

Return type folium.Map

st_last_click(*st_component*)
Get the last click feature of the map.

Parameters **st_folium** (*streamlit-folium*) – The streamlit component.

Returns The last click of the map.

Return type str

st_last_draw(*st_component*)
Get the last draw feature of the map.

Parameters **st_folium** (*streamlit-folium*) – The streamlit component.

Returns The last draw of the map.

Return type str

st_map_bounds(*st_component*)
Get the bounds of the map in the format of (miny, minx, maxy, maxx).

Parameters **st_folium** (*streamlit-folium*) – The streamlit component.

Returns The bounds of the map.

Return type tuple

st_map_center(*st_component*)
Get the center of the map.

Parameters **st_folium** (*streamlit-folium*) – The streamlit component.

Returns The center of the map.

Return type tuple

to_gradio(*width='100%'*, *height='500px'*, ***kwargs*)

Converts the map to an HTML string that can be used in Gradio. Removes unsupported elements, such as attribution and any code blocks containing functions. See https://github.com/gradio-app/gradio/issues/3190

Parameters

- **width** (*str, optional*) – The width of the map. Defaults to '100%'.
- **height** (*str, optional*) – The height of the map. Defaults to '500px'.

Returns The HTML string to use in Gradio.

Return type str

to_html(*filename=None*, ***kwargs*)
Exports a map as an HTML file.

Parameters **filename** (*str, optional*) – File path to the output HTML. Defaults to None.

Raises ValueError – If it is an invalid HTML file.

Returns A string containing the HTML code.

Return type str

to_streamlit(*width=None*, *height=600*, *scrolling=False*, *add_layer_control=True*, *bidirectional=False*, ***kwargs*)

Renders *folium.Figure* or *folium.Map* in a Streamlit app. This method is a static Streamlit Component, meaning, no information is passed back from Leaflet on browser interaction.

Parameters

- **width** (*int, optional*) – Width of the map. Defaults to None.
- **height** (*int, optional*) – Height of the map. Defaults to 600.
- **scrolling** (*bool, optional*) – Whether to allow the map to scroll. Defaults to False.
- **add_layer_control** (*bool, optional*) – Whether to add the layer control. Defaults to True.
- **bidirectional** (*bool, optional*) – Whether to add bidirectional functionality to the map. The streamlit-folium package is required to use the bidirectional functionality. Defaults to False.

Raises ImportError – If streamlit is not installed.

Returns components.html object.

Return type streamlit.components

ts_inspector(*left_ts*, *right_ts*, *left_names*, *right_names*, *left_vis={}*, *right_vis={}*, *width='130px'*, ***kwargs*)

zoom_to_bounds(*bounds*)

Zooms to a bounding box in the form of [minx, miny, maxx, maxy].

Parameters **bounds** (*list | tuple*) – A list/tuple containing minx, miny, maxx, maxy values for the bounds.

zoom_to_gdf(*gdf*)

Zooms to the bounding box of a GeoPandas GeoDataFrame.

Parameters **gdf** (*GeoDataFrame*) – A GeoPandas GeoDataFrame.

class geemap.foliumap.**SideBySideLayers**(*layer_left*, *layer_right*)

Bases: folium.elements.JSCSSMixin, folium.map.Layer

Creates a SideBySideLayers that takes two Layers and adds a sliding control with the leaflet-side-by-side plugin. Uses the Leaflet leaflet-side-by-side plugin https://github.com/digidem/leaflet-side-by-side. Adopted from https://github.com/python-visualization/folium/pull/1292/files. :param layer_left: The left Layer within the side by side control.

Must be created and added to the map before being passed to this class.

Parameters **layer_right** (*Layer.*) – The right Layer within the side by side control. Must be created and added to the map before being passed to this class.

Examples

```
>>> sidebyside = SideBySideLayers(layer_left, layer_right)
>>> sidebyside.add_to(m)
```

default_js = [('leaflet.sidebyside', 'https://cdn.jsdelivr.net/gh/digidem/leaflet-side

class geemap.foliumap.**SplitControl**(*layer_left*, *layer_right*, *name=None*, *overlay=True*, *control=False*, *show=True*)

Bases: folium.map.Layer

Creates a SplitControl that takes two Layers and adds a sliding control with the leaflet-side-by-side plugin. Uses the Leaflet leaflet-side-by-side plugin https://github.com/digidem/leaflet-side-by-side Parameters. The source code is adapted from https://github.com/python-visualization/folium/pull/1292 ———- layer_left: Layer.

The left Layer within the side by side control. Must be created and added to the map before being passed to this class.

layer_right: Layer. The right Layer within the side by side control. Must be created and added to the map before being passed to this class.

name [string, default None] The name of the Layer, as it will appear in LayerControls.

overlay [bool, default True] Adds the layer as an optional overlay (True) or the base layer (False).

control [bool, default True] Whether the Layer will be included in LayerControls.

show: bool, default True Whether the layer will be shown on opening (only for overlays).

Examples

```
>>> sidebyside = SideBySideLayers(layer_left, layer_right)
>>> sidebyside.add_to(m)
```

render(***kwargs*)

Renders the HTML representation of the element.

geemap.foliumap.**delete_dp_report**(*name*)

Deletes a datapane report.

Parameters **name** (*str*) – Name of the report to delete.

geemap.foliumap.**delete_dp_reports**()

Deletes all datapane reports.

geemap.foliumap.**ee_tile_layer**(*ee_object*, *vis_params={}*, *name='Layer untitled'*, *shown=True*, *opacity=1.0*, ***kwargs*)

Converts and Earth Engine layer to ipyleaflet TileLayer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer untitled'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

geemap.foliumap.**linked_maps**(*rows=2*, *cols=2*, *height='400px'*, *ee_objects=[]*, *vis_params=[]*, *labels=[]*, *label_position='topright'*, ***kwargs*)

geemap.foliumap.**st_map_center**(*lat*, *lon*)

Returns the map center coordinates for a given latitude and longitude. If the system variable 'map_center' exists, it is used. Otherwise, the default is returned.

Parameters

- **lat** (*float*) – Latitude.
- **lon** (*float*) – Longitude.

Raises Exception – If streamlit is not installed.

Returns The map center coordinates.

Return type list

geemap.foliumap.**st_save_bounds**(*st_component*)
Saves the map bounds to the session state.

Parameters **map** (*folium.folium.Map*) – The map to save the bounds from.

4.1.6 geemap.geemap module

Main module for interactive mapping using Google Earth Engine Python API and ipyleaflet. Keep in mind that Earth Engine functions use both camel case and snake case, such as setOptions(), setCenter(), centerObject(), addLayer(). ipyleaflet functions use snake case, such as add_tile_layer(), add_wms_layer(), add_minimap().

class geemap.geemap.**ImageOverlay**(***kwargs*)
Bases: ipyleaflet.leaflet.ImageOverlay

ImageOverlay class.

Parameters

- **url** (*str*) – http URL or local file path to the image.
- **bounds** (*tuple*) – bounding box of the image in the format of (lower_left(lat, lon), upper_right(lat, lon)), such as ((13, -130), (32, -100)).
- **name** (*str*) – The name of the layer.

class geemap.geemap.**Map**(***kwargs*)
Bases: ipyleaflet.leaflet.Map

The Map class inherits from ipyleaflet.Map. The arguments you can pass to the Map can be found at https://ipyleaflet.readthedocs.io/en/latest/map_and_basemaps/map.html. By default, the Map will add Google Maps as the basemap. Set add_google_map = False to use OpenStreetMap as the basemap.

Returns ipyleaflet map object.

Return type object

addLayer(*ee_object*, *vis_params={}*, *name=None*, *shown=True*, *opacity=1.0*)
Adds a given EE object to the map as a layer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer N'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

addLayerControl()
Adds the layer control to the map.

add_basemap(*basemap='HYBRID'*)
Adds a basemap to the map.

Parameters **basemap** (*str, optional*) – Can be one of string from basemaps. Defaults to 'HYBRID'.

add_census_data(*wms*, *layer*, *census_dict=None*, ***kwargs*)
Adds a census data layer to the map.

Parameters

- **wms** (*str*) – The wms to use. For example, "Current", "ACS 2021", "Census 2020". See the complete list at https://tigerweb.geo.census.gov/tigerwebmain/TIGERweb_wms.html
- **layer** (*str*) – The layer name to add to the map.
- **census_dict** (*dict, optional*) – A dictionary containing census data. Defaults to None. It can be obtained from the get_census_dict() function.

add_circle_markers_from_xy(*data*, *x='longitude'*, *y='latitude'*, *radius=10*, *popup=None*, ***kwargs*)
Adds a marker cluster to the map. For a list of options, see https://ipyleaflet.readthedocs.io/en/latest/api_reference/circle_marker.html

Parameters

- **data** (*str | pd.DataFrame*) – A csv or Pandas DataFrame containing x, y, z values.
- **x** (*str, optional*) – The column name for the x values. Defaults to "longitude".
- **y** (*str, optional*) – The column name for the y values. Defaults to "latitude".
- **radius** (*int, optional*) – The radius of the circle. Defaults to 10.
- **popup** (*list, optional*) – A list of column names to be used as the popup. Defaults to None.

add_cog_layer(*url*, *name='Untitled'*, *attribution=''*, *opacity=1.0*, *shown=True*, *bands=None*, *titiler_endpoint=None*, ***kwargs*)
Adds a COG TileLayer to the map.

Parameters

- **url** (*str*) – The URL of the COG tile layer.
- **name** (*str, optional*) – The layer name to use for the layer. Defaults to 'Untitled'.
- **attribution** (*str, optional*) – The attribution to use. Defaults to ''.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **bands** (*list, optional*) – A list of bands to use for the layer. Defaults to None.
- **titiler_endpoint** (*str, optional*) – Titiler endpoint. Defaults to "https://titiler.xyz".
- ****kwargs** – Arbitrary keyword arguments, including bidx, expression, nodata, unscale, resampling, rescale, color_formula, colormap, colormap_name, return_mask. See https://developmentseed.org/titiler/endpoints/cog/ and https://cogeotiff.github.io/rio-tiler/colormap/. To select a certain bands, use bidx=[1, 2, 3]

add_cog_mosaic(***kwargs*)

add_colorbar(*vis_params=None*, *cmap='gray'*, *discrete=False*, *label=None*, *orientation='horizontal'*, *position='bottomright'*, *transparent_bg=False*, *layer_name=None*, *font_size=9*, *axis_off=False*, ***kwargs*)

Add a matplotlib colorbar to the map

Parameters

- **vis_params** (*dict*) – Visualization parameters as a dictionary. See https://developers.google.com/earth-engine/guides/image_visualization for options.
- **cmap** (*str, optional*) – Matplotlib colormap. Defaults to "gray". See https://matplotlib.org/3.3.4/tutorials/colors/colormaps.html#sphx-glr-tutorials-colors-colormaps-py for options.
- **discrete** (*bool, optional*) – Whether to create a discrete colorbar. Defaults to False.
- **label** (*str, optional*) – Label for the colorbar. Defaults to None.
- **orientation** (*str, optional*) – Orientation of the colorbar, such as "vertical" and "horizontal". Defaults to "horizontal".
- **position** (*str, optional*) – Position of the colorbar on the map. It can be one of: topleft, topright, bottomleft, and bottomright. Defaults to "bottomright".
- **transparent_bg** (*bool, optional*) – Whether to use transparent background. Defaults to False.
- **layer_name** (*str, optional*) – The layer name associated with the colorbar. Defaults to None.
- **font_size** (*int, optional*) – Font size for the colorbar. Defaults to 9.
- **axis_off** (*bool, optional*) – Whether to turn off the axis. Defaults to False.

Raises

- TypeError – If the vis_params is not a dictionary.
- ValueError – If the orientation is not either horizontal or vertical.
- ValueError – If the provided min value is not scalar type.
- ValueError – If the provided max value is not scalar type.
- ValueError – If the provided opacity value is not scalar type.
- ValueError – If cmap or palette is not provided.

add_colorbar_branca(*colors*, *vmin=0*, *vmax=1.0*, *index=None*, *caption=''*, *categorical=False*, *step=None*, *height='45px'*, *transparent_bg=False*, *position='bottomright'*, *layer_name=None*, ***kwargs*)

Add a branca colorbar to the map.

Parameters

- **colors** (*list*) – The set of colors to be used for interpolation. Colors can be provided in the form: * tuples of RGBA ints between 0 and 255 (e.g: (255, 255, 0) or (255, 255, 0, 255)) * tuples of RGBA floats between 0. and 1. (e.g: (1.,1.,0.) or (1., 1., 0., 1.)) * HTML-like string (e.g: "#ffff00) * a color name or shortcut (e.g: "y" or "yellow")
- **vmin** (*int, optional*) – The minimal value for the colormap. Values lower than vmin will be bound directly to colors[0].. Defaults to 0.

- **vmax** (*float, optional*) – The maximal value for the colormap. Values higher than vmax will be bound directly to colors[-1]. Defaults to 1.0.
- **index** (*list, optional*) – The values corresponding to each color. It has to be sorted, and have the same length as colors. If None, a regular grid between vmin and vmax is created.. Defaults to None.
- **caption** (*str, optional*) – The caption for the colormap. Defaults to "".
- **categorical** (*bool, optional*) – Whether or not to create a categorical colormap. Defaults to False.
- **step** (*int, optional*) – The step to split the LinearColormap into a StepColormap. Defaults to None.
- **height** (*str, optional*) – The height of the colormap widget. Defaults to "45px".
- **transparent_bg** (*bool, optional*) – Whether to use transparent background for the colormap widget. Defaults to True.
- **position** (*str, optional*) – The position for the colormap widget. Defaults to "bottomright".
- **layer_name** (*str, optional*) – Layer name of the colorbar to be associated with. Defaults to None.

add_data(*data*, *column*, *colors=None*, *labels=None*, *cmap=None*, *scheme='Quantiles'*, *k=5*, *add_legend=True*, *legend_title=None*, *legend_kwds=None*, *classification_kwds=None*, *layer_name='Untitled'*, *style=None*, *hover_style=None*, *style_callback=None*, *info_mode='on_hover'*, *encoding='utf-8'*, ***kwargs*)

Add vector data to the map with a variety of classification schemes.

Parameters

- **data** (*str | pd.DataFrame | gpd.GeoDataFrame*) – The data to classify. It can be a filepath to a vector dataset, a pandas dataframe, or a geopandas geodataframe.
- **column** (*str*) – The column to classify.
- **cmap** (*str, optional*) – The name of a colormap recognized by matplotlib. Defaults to None.
- **colors** (*list, optional*) – A list of colors to use for the classification. Defaults to None.
- **labels** (*list, optional*) – A list of labels to use for the legend. Defaults to None.
- **scheme** (*str, optional*) – Name of a choropleth classification scheme (requires mapclassify). Name of a choropleth classification scheme (requires mapclassify). A mapclassify.MapClassifier object will be used under the hood. Supported are all schemes provided by mapclassify (e.g. 'BoxPlot', 'EqualInterval', 'FisherJenks', 'FisherJenksSampled', 'HeadTailBreaks', 'JenksCaspall', 'JenksCaspallForced', 'JenksCaspallSampled', 'MaxP', 'MaximumBreaks', 'NaturalBreaks', 'Quantiles', 'Percentiles', 'StdMean', 'UserDefined'). Arguments can be passed in classification_kwds.
- **k** (*int, optional*) – Number of classes (ignored if scheme is None or if column is categorical). Default to 5.
- **legend_kwds** (*dict, optional*) – Keyword arguments to pass to `matplotlib.pyplot.legend()` or *matplotlib.pyplot.colorbar*. Defaults to None. Keyword arguments to pass to `matplotlib.pyplot.legend()` or Additional accepted keywords when *scheme* is specified: fmt : string

A formatting specification for the bin edges of the classes in the legend. For example, to have no decimals: `{"fmt": "{:.0f}"}`.

labels [list-like] A list of legend labels to override the auto-generated labblels. Needs to have the same number of elements as the number of classes (*k*).

interval [boolean (default False)] An option to control brackets from mapclassify legend. If True, open/closed interval brackets are shown in the legend.

- **classification_kwds** (*dict, optional*) – Keyword arguments to pass to mapclassify. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to None. style is a dictionary of the following form:

  style = { "stroke": False, "color": "#ff0000", "weight": 1, "opacity": 1, "fill": True, "fillColor": "#ffffff", "fillOpacity": 1.0, "dashArray": "9" "clickable": True,

  }
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}. hover_style is a dictionary of the following form:

  hover_style = {"weight": style["weight"] + 1, "fillOpacity": 0.5}
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None. style_callback is a function that takes the feature as argument and should return a dictionary of the following form: style_callback = lambda feat: {"fillColor": feat["properties"]["color"]}
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding of the GeoJSON file. Defaults to "utf-8".

add_ee_layer(*ee_object*, *vis_params={}*, *name=None*, *shown=True*, *opacity=1.0*)

Adds a given EE object to the map as a layer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer N'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

add_gdf(*gdf, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', zoom_to_layer=True, encoding='utf-8'*)

Adds a GeoDataFrame to the map.

Parameters

- **gdf** (*GeoDataFrame*) – A GeoPandas GeoDataFrame.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **zoom_to_layer** (*bool, optional*) – Whether to zoom to the layer.
- **encoding** (*str, optional*) – The encoding of the GeoDataFrame. Defaults to "utf-8".

add_gdf_from_postgis(*sql, con, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', zoom_to_layer=True, **kwargs*)

Reads a PostGIS database and returns data as a GeoDataFrame to be added to the map.

Parameters

- **sql** (*str*) – SQL query to execute in selecting entries from database, or name of the table to read from the database.
- **con** (*sqlalchemy.engine.Engine*) – Active connection to the database to query.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **zoom_to_layer** (*bool, optional*) – Whether to zoom to the layer.

add_geojson(*in_geojson, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', encoding='utf-8'*)

Adds a GeoJSON file to the map.

Parameters

- **in_geojson** (*str | dict*) – The file path or http URL to the input GeoJSON or a dictionary containing the geojson.

- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding of the GeoJSON file. Defaults to "utf-8".

Raises `FileNotFoundError` – The provided GeoJSON file could not be found.

add_heatmap(*data*, *latitude='latitude'*, *longitude='longitude'*, *value='value'*, *name='Heat map'*, *radius=25*, ***kwargs*)

Adds a heat map to the map. Reference: https://ipyleaflet.readthedocs.io/en/latest/api_reference/heatmap.html

Parameters

- **data** (*str | list | pd.DataFrame*) – File path or HTTP URL to the input file or a list of data points in the format of [[x1, y1, z1], [x2, y2, z2]]. For example, https://raw.githubusercontent.com/giswqs/leafmap/master/examples/data/world_cities.csv
- **latitude** (*str, optional*) – The column name of latitude. Defaults to "latitude".
- **longitude** (*str, optional*) – The column name of longitude. Defaults to "longitude".
- **value** (*str, optional*) – The column name of values. Defaults to "value".
- **name** (*str, optional*) – Layer name to use. Defaults to "Heat map".
- **radius** (*int, optional*) – Radius of each "point" of the heatmap. Defaults to 25.

Raises `ValueError` – If data is not a list.

add_html(*html*, *position='bottomright'*, ***kwargs*)

Add HTML to the map.

Parameters

- **html** (*str*) – The HTML to add.
- **position** (*str, optional*) – The position of the HTML, can be one of "topleft", "topright", "bottomleft", "bottomright". Defaults to "bottomright".

add_image(*image*, *position='bottomright'*, ***kwargs*)

Add an image to the map.

Parameters

- **image** (*str | ipywidgets.Image*) – The image to add.
- **position** (*str, optional*) – The position of the image, can be one of "topleft", "topright", "bottomleft", "bottomright". Defaults to "bottomright".

add_kml(*in_kml, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds a GeoJSON file to the map.

Parameters

- **in_kml** (*str*) – The input file path to the KML.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

Raises FileNotFoundError – The provided KML file could not be found.

add_labels(*data, column, font_size='12pt', font_color='black', font_family='arial', font_weight='normal', x='longitude', y='latitude', draggable=True, layer_name='Labels', **kwargs*)

Adds a label layer to the map. Reference: https://ipyleaflet.readthedocs.io/en/latest/api_reference/divicon.html

Parameters

- **data** (*pd.DataFrame | ee.FeatureCollection*) – The input data to label.
- **column** (*str*) – The column name of the data to label.
- **font_size** (*str, optional*) – The font size of the labels. Defaults to "12pt".
- **font_color** (*str, optional*) – The font color of the labels. Defaults to "black".
- **font_family** (*str, optional*) – The font family of the labels. Defaults to "arial".
- **font_weight** (*str, optional*) – The font weight of the labels, can be normal, bold. Defaults to "normal".
- **x** (*str, optional*) – The column name of the longitude. Defaults to "longitude".
- **y** (*str, optional*) – The column name of the latitude. Defaults to "latitude".
- **draggable** (*bool, optional*) – Whether the labels are draggable. Defaults to True.
- **layer_name** (*str, optional*) – Layer name to use. Defaults to "Labels".

add_landsat_ts_gif(*layer_name='Timelapse', roi=None, label=None, start_year=1984, end_year=2021, start_date='06-10', end_date='09-20', bands=['NIR', 'Red', 'Green'], vis_params=None, dimensions=768, frames_per_second=10, font_size=30, font_color='white', add_progress_bar=True, progress_bar_color='white', progress_bar_height=5, out_gif=None, download=False, apply_fmask=True, nd_bands=None, nd_threshold=0, nd_palette=['black', 'blue']*)

Adds a Landsat timelapse to the map.

Parameters

- **layer_name** (*str, optional*) – Layer name to show under the layer control. Defaults to 'Timelapse'.
- **roi** (*object, optional*) – Region of interest to create the timelapse. Defaults to None.
- **label** (*str, optional*) – A label to show on the GIF, such as place name. Defaults to None.
- **start_year** (*int, optional*) – Starting year for the timelapse. Defaults to 1984.
- **end_year** (*int, optional*) – Ending year for the timelapse. Defaults to 2021.
- **start_date** (*str, optional*) – Starting date (month-day) each year for filtering ImageCollection. Defaults to '06-10'.
- **end_date** (*str, optional*) – Ending date (month-day) each year for filtering ImageCollection. Defaults to '09-20'.
- **bands** (*list, optional*) – Three bands selected from ['Blue', 'Green', 'Red', 'NIR', 'SWIR1', 'SWIR2', 'pixel_qa']. Defaults to ['NIR', 'Red', 'Green'].
- **vis_params** (*dict, optional*) – Visualization parameters. Defaults to None.
- **dimensions** (*int, optional*) – a number or pair of numbers in format WIDTHxHEIGHT) Maximum dimensions of the thumbnail to render, in pixels. If only one number is passed, it is used as the maximum, and the other dimension is computed by proportional scaling. Defaults to 768.
- **frames_per_second** (*int, optional*) – Animation speed. Defaults to 10.
- **font_size** (*int, optional*) – Font size of the animated text and label. Defaults to 30.
- **font_color** (*str, optional*) – Font color of the animated text and label. Defaults to 'black'.
- **add_progress_bar** (*bool, optional*) – Whether to add a progress bar at the bottom of the GIF. Defaults to True.
- **progress_bar_color** (*str, optional*) – Color for the progress bar. Defaults to 'white'.
- **progress_bar_height** (*int, optional*) – Height of the progress bar. Defaults to 5.
- **out_gif** (*str, optional*) – File path to the output animated GIF. Defaults to None.
- **download** (*bool, optional*) – Whether to download the gif. Defaults to False.
- **apply_fmask** (*bool, optional*) – Whether to apply Fmask (Function of mask) for automated clouds, cloud shadows, snow, and water masking.
- **nd_bands** (*list, optional*) – A list of names specifying the bands to use, e.g., ['Green', 'SWIR1']. The normalized difference is computed as (first second) / (first + second). Note that negative input values are forced to 0 so that the result is confined to the range (-1, 1).
- **nd_threshold** (*float, optional*) – The threshold for extacting pixels from the normalized difference band.

- **nd_palette** (*str, optional*) – The color palette to use for displaying the normalized difference band.

add_layer_control()
Adds the layer control to the map.

add_legend(*title='Legend'*, *legend_dict=None*, *labels=None*, *colors=None*, *position='bottomright'*, *builtin_legend=None*, *layer_name=None*, ***kwargs*)
Adds a customized basemap to the map.

Parameters

- **title** (*str, optional*) – Title of the legend. Defaults to 'Legend'.
- **legend_dict** (*dict, optional*) – A dictionary containing legend items as keys and color as values. If provided, legend_keys and legend_colors will be ignored. Defaults to None.
- **labels** (*list, optional*) – A list of legend keys. Defaults to None.
- **colors** (*list, optional*) – A list of legend colors. Defaults to None.
- **position** (*str, optional*) – Position of the legend. Defaults to 'bottomright'.
- **builtin_legend** (*str, optional*) – Name of the builtin legend to add to the map. Defaults to None.
- **layer_name** (*str, optional*) – Layer name of the legend to be associated with. Defaults to None.

add_local_tile(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name=None*, ***kwargs*)
Add a local raster dataset to the map.

If you are using this function in JupyterHub on a remote server and the raster does not render properly, try running the following two lines before calling this function:

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = 'proxy/{port}'

Parameters

- **source** (*str*) – The path to the GeoTIFF file or the URL of the Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to None.

add_marker(*location*, ***kwargs*)
Adds a marker to the map. More info about marker at https://ipyleaflet.readthedocs.io/en/latest/api_reference/marker.html.

Parameters

- **location** (*list | tuple*) – The location of the marker in the format of [lat, lng].
- ****kwargs** – Keyword arguments for the marker.

add_marker_cluster(*event='click'*, *add_marker=True*)
Captures user inputs and add markers to the map.

Parameters

- **event** (*str, optional*) – [description]. Defaults to 'click'.
- **add_marker** (*bool, optional*) – If True, add markers to the map. Defaults to True.

Returns a marker cluster.

Return type object

add_minimap(*zoom=5*, *position='bottomright'*)
Adds a minimap (overview) to the ipyleaflet map.

Parameters

- **zoom** (*int, optional*) – Initial map zoom level. Defaults to 5.
- **position** (*str, optional*) – Position of the minimap. Defaults to "bottomright".

add_netcdf(*filename*, *variables=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name='NetCDF layer'*, *shift_lon=True*, *lat='lat'*, *lon='lon'*, ***kwargs*)
Generate an ipyleaflet/folium TileLayer from a netCDF file. If you are using this function in JupyterHub on a remote server (e.g., Binder, Microsoft Planetary Computer), try adding to following two lines to the beginning of the notebook if the raster does not render properly.

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = f'{os.environ['JUPYTERHUB_SERVICE_PREFIX'].lstrip('/')}/proxy/{{port}}'

Parameters

- **filename** (*str*) – File path or HTTP URL to the netCDF file.
- **variables** (*int, optional*) – The variable/band names to extract data from the netCDF file. Defaults to None. If None, all variables will be extracted.
- **port** (*str, optional*) – The port to use for the server. Defaults to "default".
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.

- **layer_name** (*str, optional*) – The layer name to use. Defaults to "netCDF layer".
- **shift_lon** (*bool, optional*) – Flag to shift longitude values from [0, 360] to the range [-180, 180]. Defaults to True.
- **lat** (*str, optional*) – Name of the latitude variable. Defaults to 'lat'.
- **lon** (*str, optional*) – Name of the longitude variable. Defaults to 'lon'.

add_osm(*query, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', which_result=None, by_osmid=False, buffer_dist=None, to_ee=False, geodesic=True*)

Adds OSM data to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **which_result** (*INT, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **by_osmid** (*bool, optional*) – If True, handle query as an OSM ID for lookup rather than text search. Defaults to False.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.
- **to_ee** (*bool, optional*) – Whether to convert the csv to an ee.FeatureCollection.
- **geodesic** (*bool, optional*) – Whether line segments should be interpreted as spherical geodesics. If false, indicates that line segments should be interpreted as planar lines in the specified CRS. If absent, defaults to true if the CRS is geographic (including the default EPSG:4326), or to false if the CRS is projected.

add_osm_from_address(*address, tags, dist=1000, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within some distance N, S, E, W of address to the map.

Parameters

- **address** (*str*) – The address to geocode and use as the central point around which to get the geometries.

- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **dist** (*int, optional*) – Distance in meters. Defaults to 1000.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_bbox(*north, south, east, west, tags, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within a N, S, E, W bounding box to the map.

Parameters

- **north** (*float*) – Northern latitude of bounding box.
- **south** (*float*) – Southern latitude of bounding box.
- **east** (*float*) – Eastern longitude of bounding box.
- **west** (*float*) – Western longitude of bounding box.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.

- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_geocode(*query, which_result=None, by_osmid=False, buffer_dist=None, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM data of place(s) by name or ID to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **which_result** (*int, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **by_osmid** (*bool, optional*) – If True, handle query as an OSM ID for lookup rather than text search. Defaults to False.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_place(*query, tags, which_result=None, buffer_dist=None, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within boundaries of geocodable place(s) to the map.

Parameters

- **query** (*str | dict | list*) – Query string(s) or structured dict(s) to geocode.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.

- **which_result** (*int, optional*) – Which geocoding result to use. if None, auto-select the first (Multi)Polygon or raise an error if OSM doesn't return one. to get the top match regardless of geometry type, set which_result=1. Defaults to None.
- **buffer_dist** (*float, optional*) – Distance to buffer around the place geometry, in meters. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_point(*center_point, tags, dist=1000, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within some distance N, S, E, W of a point to the map.

Parameters

- **center_point** (*tuple*) – The (lat, lng) center point around which to get the geometries.
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **dist** (*int, optional*) – Distance in meters. Defaults to 1000.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_polygon(*polygon, tags, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within boundaries of a (multi)polygon to the map.

Parameters

- **polygon** (*shapely.geometry.Polygon | shapely.geometry.MultiPolygon*) – Geographic boundaries to fetch geometries within
- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_osm_from_view(*tags, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover'*)

Adds OSM entities within the current map view to the map.

Parameters

- **tags** (*dict*) – Dict of tags used for finding objects in the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one given tag. The dict keys should be OSM tags, (e.g., building, landuse, highway, etc) and the dict values should be either True to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, tags = {'building': True} would return all building footprints in the area. tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'} would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.

- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".

add_planet_by_month(*year=2016*, *month=1*, *name=None*, *api_key=None*, *token_name='PLANET_API_KEY'*)

Adds a Planet global mosaic by month to the map. To get a Planet API key, see https://developers.planet.com/quickstart/apis

Parameters

- **year** (*int, optional*) – The year of Planet global mosaic, must be >=2016. Defaults to 2016.
- **month** (*int, optional*) – The month of Planet global mosaic, must be 1-12. Defaults to 1.
- **name** (*str, optional*) – The layer name to use. Defaults to None.
- **api_key** (*str, optional*) – The Planet API key. Defaults to None.
- **token_name** (*str, optional*) – The environment variable name of the API key. Defaults to "PLANET_API_KEY".

add_planet_by_quarter(*year=2016*, *quarter=1*, *name=None*, *api_key=None*, *token_name='PLANET_API_KEY'*)

Adds a Planet global mosaic by quarter to the map. To get a Planet API key, see https://developers.planet.com/quickstart/apis

Parameters

- **year** (*int, optional*) – The year of Planet global mosaic, must be >=2016. Defaults to 2016.
- **quarter** (*int, optional*) – The quarter of Planet global mosaic, must be 1-12. Defaults to 1.
- **name** (*str, optional*) – The layer name to use. Defaults to None.
- **api_key** (*str, optional*) – The Planet API key. Defaults to None.
- **token_name** (*str, optional*) – The environment variable name of the API key. Defaults to "PLANET_API_KEY".

add_point_layer(*filename*, *popup=None*, *layer_name='Marker Cluster'*, ***kwargs*)

Adds a point layer to the map with a popup attribute.

Parameters

- **filename** (*str*) – str, http url, path object or file-like object. Either the absolute or relative path to the file or URL to be opened, or any object with a read() method (such as an open file or StringIO)
- **popup** (*str | list, optional*) – Column name(s) to be used for popup. Defaults to None.
- **layer_name** (*str, optional*) – A layer name to use. Defaults to "Marker Cluster".

Raises

- `ValueError` – If the specified column name does not exist.
- `ValueError` – If the specified column names do not exist.

add_points_from_xy(*data, x='longitude', y='latitude', popup=None, layer_name='Marker Cluster', color_column=None, marker_colors=None, icon_colors=['white'], icon_names=['info'], spin=False, add_legend=True, **kwargs*)

Adds a marker cluster to the map.

Parameters

- **data** (*str | pd.DataFrame*) – A csv or Pandas DataFrame containing x, y, z values.
- **x** (*str, optional*) – The column name for the x values. Defaults to "longitude".
- **y** (*str, optional*) – The column name for the y values. Defaults to "latitude".
- **popup** (*list, optional*) – A list of column names to be used as the popup. Defaults to None.
- **layer_name** (*str, optional*) – The name of the layer. Defaults to "Marker Cluster".
- **color_column** (*str, optional*) – The column name for the color values. Defaults to None.
- **marker_colors** (*list, optional*) – A list of colors to be used for the markers. Defaults to None.
- **icon_colors** (*list, optional*) – A list of colors to be used for the icons. Defaults to ['white'].
- **icon_names** (*list, optional*) – A list of names to be used for the icons. More icons can be found at https://fontawesome.com/v4/icons. Defaults to ['info'].
- **spin** (*bool, optional*) – If True, the icon will spin. Defaults to False.
- **add_legend** (*bool, optional*) – If True, a legend will be added to the map. Defaults to True.

add_raster(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name=None*, ***kwargs*)

Add a local raster dataset to the map.

If you are using this function in JupyterHub on a remote server and the raster does not render properly, try running the following two lines before calling this function:

import os os.environ['LOCALTILESERVER_CLIENT_PREFIX'] = 'proxy/{port}'

Parameters

- **source** (*str*) – The path to the GeoTIFF file or the URL of the Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.

- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to None.

add_raster_legacy(*image*, *bands=None*, *layer_name=None*, *colormap=None*, *x_dim='x'*, *y_dim='y'*)

Adds a local raster dataset to the map.

Parameters

- **image** (*str*) – The image file path.
- **bands** (*int or list, optional*) – The image bands to use. It can be either a number (e.g., 1) or a list (e.g., [3, 2, 1]). Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use for the raster. Defaults to None.
- **colormap** (*str, optional*) – The name of the colormap to use for the raster, such as 'gray' and 'terrain'. More can be found at https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html. Defaults to None.
- **x_dim** (*str, optional*) – The x dimension. Defaults to 'x'.
- **y_dim** (*str, optional*) – The y dimension. Defaults to 'y'.

add_remote_tile(*source*, *band=None*, *palette=None*, *vmin=None*, *vmax=None*, *nodata=None*, *attribution=None*, *layer_name=None*, ***kwargs*)

Add a remote Cloud Optimized GeoTIFF (COG) to the map.

Parameters

- **source** (*str*) – The path to the remote Cloud Optimized GeoTIFF.
- **band** (*int, optional*) – The band to use. Band indexing starts at 1. Defaults to None.
- **palette** (*str, optional*) – The name of the color palette from *palettable* to use when plotting a single band. See https://jiffyclub.github.io/palettable. Default is greyscale
- **vmin** (*float, optional*) – The minimum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **vmax** (*float, optional*) – The maximum value to use when colormapping the palette when plotting a single band. Defaults to None.
- **nodata** (*float, optional*) – The value from the band to use to interpret as not valid data. Defaults to None.
- **attribution** (*str, optional*) – Attribution for the source raster. This defaults to a message about it being a local file.. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to None.

add_shapefile(*in_shp, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', encoding='utf-8'*)

Adds a shapefile to the map.

Parameters

- **in_shp** (*str*) – The input file path to the shapefile.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".

- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding of the shapefile. Defaults to "utf-8".

Raises FileNotFoundError – The provided shapefile could not be found.

add_shp(*in_shp, layer_name='Untitled', style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', encoding='utf-8'*)

Adds a shapefile to the map.

Parameters

- **in_shp** (*str*) – The input file path to the shapefile.
- **layer_name** (*str, optional*) – The layer name to be used.. Defaults to "Untitled".
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding of the shapefile. Defaults to "utf-8".

Raises FileNotFoundError – The provided shapefile could not be found.

add_stac_layer(*url=None, collection=None, item=None, assets=None, bands=None, titiler_endpoint=None, name='STAC Layer', attribution='', opacity=1.0, shown=True, **kwargs*)

Adds a STAC TileLayer to the map.

Parameters

- **url** (*str*) – HTTP URL to a STAC item, e.g., https://canada-spot-ortho.s3.amazonaws.com/canada_spot_orthoimages/canada_spot5_orthoimages/S5_2007/S5_11055_6057_20070622/S5_11055_6057_20070622.json
- **collection** (*str*) – The Microsoft Planetary Computer STAC collection ID, e.g., landsat-8-c2-l2.

- **item** (*str*) – The Microsoft Planetary Computer STAC item ID, e.g., LC08_L2SP_047027_20201204_02_T1.
- **assets** (*str | list*) – The Microsoft Planetary Computer STAC asset ID, e.g., ["SR_B7", "SR_B5", "SR_B4"].
- **bands** (*list*) – A list of band names, e.g., ["SR_B7", "SR_B5", "SR_B4"]
- **titiler_endpoint** (*str, optional*) – Titiler endpoint, e.g., "https://titiler.xyz", "https://planetarycomputer.microsoft.com/api/data/v1", "planetary-computer", "pc". Defaults to None.
- **name** (*str, optional*) – The layer name to use for the layer. Defaults to 'STAC Layer'.
- **attribution** (*str, optional*) – The attribution to use. Defaults to ''.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.

add_styled_vector(*ee_object*, *column*, *palette*, *layer_name='Untitled'*, ***kwargs*)

Adds a styled vector to the map.

Parameters

- **ee_object** (*object*) – An ee.FeatureCollection.
- **column** (*str*) – The column name to use for styling.
- **palette** (*list | dict*) – The palette (e.g., list of colors or a dict containing label and color pairs) to use for styling.
- **layer_name** (*str, optional*) – The name to be used for the new layer. Defaults to "Untitled".

add_text(*text*, *fontsize=20*, *fontcolor='black'*, *bold=False*, *padding='5px'*, *background=True*, *bg_color='white'*, *border_radius='5px'*, *position='bottomright'*, ***kwargs*)

Add text to the map.

Parameters

- **text** (*str*) – The text to add.
- **fontsize** (*int, optional*) – The font size. Defaults to 20.
- **fontcolor** (*str, optional*) – The font color. Defaults to "black".
- **bold** (*bool, optional*) – Whether to use bold font. Defaults to False.
- **padding** (*str, optional*) – The padding. Defaults to "5px".
- **background** (*bool, optional*) – Whether to use background. Defaults to True.
- **bg_color** (*str, optional*) – The background color. Defaults to "white".
- **border_radius** (*str, optional*) – The border radius. Defaults to "5px".
- **position** (*str, optional*) – The position of the widget. Defaults to "bottomright".

add_tile_layer(*url='https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'*, *name='Untitled'*, *attribution=''*, *opacity=1.0*, *shown=True*, ***kwargs*)

Adds a TileLayer to the map.

Parameters

- **url** (*str, optional*) – The URL of the tile layer. Defaults to 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'.
- **name** (*str, optional*) – The layer name to use for the layer. Defaults to 'Untitled'.
- **attribution** (*str, optional*) – The attribution to use. Defaults to ''.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.

add_time_slider(*ee_object*, *vis_params={}*, *region=None*, *layer_name='Time series'*, *labels=None*, *time_interval=1*, *position='bottomright'*, *slider_length='150px'*, *date_format='YYYY-MM-dd'*, *opacity=1.0*, ***kwargs*)

Adds a time slider to the map.

Parameters

- **ee_object** (*ee.Image | ee.ImageCollection*) – The Image or ImageCollection to visualize.
- **vis_params** (*dict, optional*) – Visualization parameters to use for visualizing image. Defaults to {}.
- **region** (*ee.Geometry | ee.FeatureCollection*) – The region to visualize.
- **layer_name** (*str, optional*) – The layer name to be used. Defaults to "Time series".
- **labels** (*list, optional*) – The list of labels to be used for the time series. Defaults to None.
- **time_interval** (*int, optional*) – Time interval in seconds. Defaults to 1.
- **position** (*str, optional*) – Position to place the time slider, can be any of ['topleft', 'topright', 'bottomleft', 'bottomright']. Defaults to "bottomright".
- **slider_length** (*str, optional*) – Length of the time slider. Defaults to "150px".
- **date_format** (*str, optional*) – The date format to use. Defaults to 'YYYY-MM-dd'.
- **opacity** (*float, optional*) – The opacity of layers. Defaults to 1.0.

Raises TypeError – If the ee_object is not ee.Image | ee.ImageCollection.

add_vector(*filename, layer_name='Untitled', to_ee=False, bbox=None, mask=None, rows=None, style={}, hover_style={}, style_callback=None, fill_colors=['black'], info_mode='on_hover', encoding='utf-8', **kwargs*)

Adds any geopandas-supported vector dataset to the map.

Parameters

- **filename** (*str*) – Either the absolute or relative path to the file or URL to be opened, or any object with a read() method (such as an open file or StringIO).
- **layer_name** (*str, optional*) – The layer name to use. Defaults to "Untitled".
- **to_ee** (*bool, optional*) – Whether to convert the GeoJSON to ee.FeatureCollection. Defaults to False.
- **bbox** (*tuple | GeoDataFrame or GeoSeries | shapely Geometry, optional*) – Filter features by given bounding box, GeoSeries, GeoDataFrame or a shapely geometry. CRS mis-matches are resolved if given a GeoSeries or GeoDataFrame. Cannot be used with mask. Defaults to None.

- **mask** (*dict | GeoDataFrame or GeoSeries | shapely Geometry, optional*) – Filter for features that intersect with the given dict-like geojson geometry, GeoSeries, GeoDataFrame or shapely geometry. CRS mis-matches are resolved if given a GeoSeries or GeoDataFrame. Cannot be used with bbox. Defaults to None.
- **rows** (*int or slice, optional*) – Load in specific rows by passing an integer (first n rows) or a slice() object.. Defaults to None.
- **style** (*dict, optional*) – A dictionary specifying the style to be used. Defaults to {}.
- **hover_style** (*dict, optional*) – Hover style dictionary. Defaults to {}.
- **style_callback** (*function, optional*) – Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument. Defaults to None.
- **fill_colors** (*list, optional*) – The random colors to use for filling polygons. Defaults to ["black"].
- **info_mode** (*str, optional*) – Displays the attributes by either on_hover or on_click. Any value other than "on_hover" or "on_click" will be treated as None. Defaults to "on_hover".
- **encoding** (*str, optional*) – The encoding to use to read the file. Defaults to "utf-8".

add_velocity(*data*, *zonal_speed*, *meridional_speed*, *latitude_dimension='lat'*, *longitude_dimension='lon'*, *level_dimension='lev'*, *level_index=0*, *time_index=0*, *velocity_scale=0.01*, *max_velocity=20*, *display_options={}*, *name='Velocity'*)

Add a velocity layer to the map.

Parameters

- **data** (*str | xr.Dataset*) – The data to use for the velocity layer. It can be a file path to a NetCDF file or an xarray Dataset.
- **zonal_speed** (*str*) – Name of the zonal speed in the dataset. See https://en.wikipedia.org/wiki/Zonal_and_meridional_flow.
- **meridional_speed** (*str*) – Name of the meridional speed in the dataset. See https://en.wikipedia.org/wiki/Zonal_and_meridional_flow.
- **latitude_dimension** (*str, optional*) – Name of the latitude dimension in the dataset. Defaults to 'lat'.
- **longitude_dimension** (*str, optional*) – Name of the longitude dimension in the dataset. Defaults to 'lon'.
- **level_dimension** (*str, optional*) – Name of the level dimension in the dataset. Defaults to 'lev'.
- **level_index** (*int, optional*) – The index of the level dimension to display. Defaults to 0.
- **time_index** (*int, optional*) – The index of the time dimension to display. Defaults to 0.
- **velocity_scale** (*float, optional*) – The scale of the velocity. Defaults to 0.01.
- **max_velocity** (*int, optional*) – The maximum velocity to display. Defaults to 20.

- **display_options** (*dict, optional*) – The display options for the velocity layer. Defaults to {}. See https://bit.ly/3uf8t6w.
- **name** (*str, optional*) – Layer name to use . Defaults to 'Velocity'.

Raises

- ImportError – If the xarray package is not installed.
- ValueError – If the data is not a NetCDF file or an xarray Dataset.

add_widget(*content*, *position='bottomright'*, ***kwargs*)
Add a widget (e.g., text, HTML, figure) to the map.

Parameters

- **content** (*str | ipywidgets.Widget | object*) – The widget to add.
- **position** (*str, optional*) – The position of the widget. Defaults to "bottomright".
- ****kwargs** – Other keyword arguments for ipywidgets.HTML().

add_wms_layer(*url*, *layers*, *name=None*, *attribution=''*, *format='image/png'*, *transparent=True*, *opacity=1.0*, *shown=True*, ***kwargs*)
Add a WMS layer to the map.

Parameters

- **url** (*str*) – The URL of the WMS web service.
- **layers** (*str*) – Comma-separated list of WMS layers to show.
- **name** (*str, optional*) – The layer name to use on the layer control. Defaults to None.
- **attribution** (*str, optional*) – The attribution of the data layer. Defaults to ''.
- **format** (*str, optional*) – WMS image format (use 'image/png' for layers with transparency). Defaults to 'image/png'.
- **transparent** (*bool, optional*) – If True, the WMS service will return images with transparency. Defaults to True.
- **opacity** (*float, optional*) – The opacity of the layer. Defaults to 1.0.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.

add_xy_data(*in_csv*, *x='longitude'*, *y='latitude'*, *label=None*, *layer_name='Marker cluster'*, *to_ee=False*)
Adds points from a CSV file containing lat/lon information and display data on the map.

Parameters

- **in_csv** (*str*) – The file path to the input CSV file.
- **x** (*str, optional*) – The name of the column containing longitude coordinates. Defaults to "longitude".
- **y** (*str, optional*) – The name of the column containing latitude coordinates. Defaults to "latitude".
- **label** (*str, optional*) – The name of the column containing label information to used for marker popup. Defaults to None.
- **layer_name** (*str, optional*) – The layer name to use. Defaults to "Marker cluster".

- **to_ee** (*bool, optional*) – Whether to convert the csv to an ee.FeatureCollection.

Raises

- FileNotFoundError – The specified input csv does not exist.
- ValueError – The specified x column does not exist.
- ValueError – The specified y column does not exist.
- ValueError – The specified label column does not exist.

add_xyz_service(*provider*, ***kwargs*)
Add a XYZ tile layer to the map.

Parameters **provider** (*str*) – A tile layer name starts with xyz or qms. For example, xyz.OpenTopoMap,

Raises ValueError – The provider is not valid. It must start with xyz or qms.

basemap_demo()
A demo for using geemap basemaps.

centerObject(*ee_object*, *zoom=None*)
Centers the map view on a given object.

Parameters

- **ee_object** (*Element|Geometry*) – An Earth Engine object to center on a geometry, image or feature.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

center_object(*ee_object*, *zoom=None*)
Centers the map view on a given object.

Parameters

- **ee_object** (*Element|Geometry*) – An Earth Engine object to center on a geometry, image or feature.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

create_vis_widget(*layer_dict*)
Create a GUI for changing layer visualization parameters interactively.

Parameters **layer_dict** (*dict*) – A dict containning information about the layer. It is an element from Map.ee_layer_dict.

Returns An ipywidget.

Return type object

draw_layer_on_top()
Move user-drawn feature layer to the top of all layers.

extract_values_to_points(*filename*)
Exports pixel values to a csv file based on user-drawn geometries.

Parameters **filename** (*str*) – The output file path to the csv file or shapefile.

find_layer(*name*)
Finds layer by name

Parameters **name** (*str*) – Name of the layer to find.

Returns ipyleaflet layer object.

Return type object

find_layer_index(*name*)
Finds layer index by name

Parameters **name** (*str*) – Name of the layer to find.

Returns Index of the layer with the specified name

Return type int

getBounds(*asGeoJSON=False*)
Returns the bounds of the current map view, as a list in the format [west, south, east, north] in degrees.

Parameters **asGeoJSON** (*bool, optional*) – If true, returns map bounds as GeoJSON. Defaults to False.

Returns A list in the format [west, south, east, north] in degrees.

Return type list | dict

getScale()
Returns the approximate pixel scale of the current map view, in meters.

Returns Map resolution in meters.

Return type float

get_bounds(*asGeoJSON=False*)
Returns the bounds of the current map view, as a list in the format [west, south, east, north] in degrees.

Parameters **asGeoJSON** (*bool, optional*) – If true, returns map bounds as GeoJSON. Defaults to False.

Returns A list in the format [west, south, east, north] in degrees.

Return type list | dict

get_scale()
Returns the approximate pixel scale of the current map view, in meters.

Returns Map resolution in meters.

Return type float

image_overlay(*url*, *bounds*, *name*)
Overlays an image from the Internet or locally on the map.

Parameters

- **url** (*str*) – http URL or local file path to the image.
- **bounds** (*tuple*) – bounding box of the image in the format of (lower_left(lat, lon), upper_right(lat, lon)), such as ((13, -130), (32, -100)).
- **name** (*str*) – name of the layer to show on the layer control.

inspector(*latlon*)
Create the Inspector GUI.

Parameters **latlon** (*list | tuple*) – The coordinates (lat, lon) of the point.

Returns The ipytree tree widget for the Inspector GUI.

Return type ipytree.Tree

layer_opacity(*name*, *opacity=1.0*)
Changes layer opacity.

Parameters

- **name** (*str*) – The name of the layer to change opacity.
- **opacity** (*float, optional*) – The opacity value to set. Defaults to 1.0.

marker_cluster()
Adds a marker cluster to the map and returns a list of ee.Feature, which can be accessed using Map.ee_marker_cluster.

Returns a list of ee.Feature

Return type object

plot(*x*, *y*, *plot_type=None*, *overlay=False*, *position='bottomright'*, *min_width=None*, *max_width=None*, *min_height=None*, *max_height=None*, ***kwargs*)
Creates a plot based on x-array and y-array data.

Parameters

- **x** (*numpy.ndarray or list*) – The x-coordinates of the plotted line.
- **y** (*numpy.ndarray or list*) – The y-coordinates of the plotted line.
- **plot_type** (*str, optional*) – The plot type can be one of "None", "bar", "scatter" or "hist". Defaults to None.
- **overlay** (*bool, optional*) – Whether to overlay plotted lines on the figure. Defaults to False.
- **position** (*str, optional*) – Position of the control, can be 'bottomleft', 'bottomright', 'topleft', or 'topright'. Defaults to 'bottomright'.
- **min_width** (*int, optional*) – Min width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_width** (*int, optional*) – Max width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **min_height** (*int, optional*) – Min height of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_height** (*int, optional*) – Max height of the widget (in pixels), if None it will respect the content size. Defaults to None.

plot_demo(*iterations=20*, *plot_type=None*, *overlay=False*, *position='bottomright'*, *min_width=None*, *max_width=None*, *min_height=None*, *max_height=None*, ***kwargs*)
A demo of interactive plotting using random pixel coordinates.

Parameters

- **iterations** (*int, optional*) – How many iterations to run for the demo. Defaults to 20.
- **plot_type** (*str, optional*) – The plot type can be one of "None", "bar", "scatter" or "hist". Defaults to None.
- **overlay** (*bool, optional*) – Whether to overlay plotted lines on the figure. Defaults to False.
- **position** (*str, optional*) – Position of the control, can be 'bottomleft', 'bottomright', 'topleft', or 'topright'. Defaults to 'bottomright'.
- **min_width** (*int, optional*) – Min width of the widget (in pixels), if None it will respect the content size. Defaults to None.

- **max_width** (*int, optional*) – Max width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **min_height** (*int, optional*) – Min height of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_height** (*int, optional*) – Max height of the widget (in pixels), if None it will respect the content size. Defaults to None.

plot_raster(*ee_object=None*, *sample_scale=None*, *plot_type=None*, *overlay=False*, *position='bottomright'*, *min_width=None*, *max_width=None*, *min_height=None*, *max_height=None*, ***kwargs*)

Interactive plotting of Earth Engine data by clicking on the map.

Parameters

- **ee_object** (*object, optional*) – The ee.Image or ee.ImageCollection to sample. Defaults to None.
- **sample_scale** (*float, optional*) – A nominal scale in meters of the projection to sample in. Defaults to None.
- **plot_type** (*str, optional*) – The plot type can be one of "None", "bar", "scatter" or "hist". Defaults to None.
- **overlay** (*bool, optional*) – Whether to overlay plotted lines on the figure. Defaults to False.
- **position** (*str, optional*) – Position of the control, can be 'bottomleft', 'bottomright', 'topleft', or 'topright'. Defaults to 'bottomright'.
- **min_width** (*int, optional*) – Min width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_width** (*int, optional*) – Max width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **min_height** (*int, optional*) – Min height of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_height** (*int, optional*) – Max height of the widget (in pixels), if None it will respect the content size. Defaults to None.

remove_colorbar()

Remove colorbar from the map.

remove_colorbars()

Remove all colorbars from the map.

remove_drawn_features()

Removes user-drawn geometries from the map

remove_ee_layer(*name*)

Removes an Earth Engine layer.

Parameters **name** (*str*) – The name of the Earth Engine layer to remove.

remove_labels()

Removes all labels from the map.

remove_last_drawn()

Removes user-drawn geometries from the map

remove_legend()

Remove legend from the map.

remove_legends()
Remove all legends from the map.

setCenter(*lon*, *lat*, *zoom=None*)
Centers the map view at a given coordinates with the given zoom level.

Parameters

- **lon** (*float*) – The longitude of the center, in degrees.
- **lat** (*float*) – The latitude of the center, in degrees.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

setControlVisibility(*layerControl=True*, *fullscreenControl=True*, *latLngPopup=True*)
Sets the visibility of the controls on the map.

Parameters

- **layerControl** (*bool, optional*) – Whether to show the control that allows the user to toggle layers on/off. Defaults to True.
- **fullscreenControl** (*bool, optional*) – Whether to show the control that allows the user to make the map full-screen. Defaults to True.
- **latLngPopup** (*bool, optional*) – Whether to show the control that pops up the Lat/lon when the user clicks on the map. Defaults to True.

setOptions(*mapTypeId='HYBRID'*, *styles=None*, *types=None*)
Adds Google basemap and controls to the ipyleaflet map.

Parameters

- **mapTypeId** (*str, optional*) – A mapTypeId to set the basemap to. Can be one of "ROADMAP", "SATELLITE", "HYBRID" or "TERRAIN" to select one of the standard Google Maps API map types. Defaults to 'HYBRID'.
- **styles** (*object, optional*) – A dictionary of custom MapTypeStyle objects keyed with a name that will appear in the map's Map Type Controls. Defaults to None.
- **types** (*list, optional*) – A list of mapTypeIds to make available. If omitted, but opt_styles is specified, appends all of the style keys to the standard Google Maps API map types.. Defaults to None.

set_center(*lon*, *lat*, *zoom=None*)
Centers the map view at a given coordinates with the given zoom level.

Parameters

- **lon** (*float*) – The longitude of the center, in degrees.
- **lat** (*float*) – The latitude of the center, in degrees.
- **zoom** (*int, optional*) – The zoom level, from 1 to 24. Defaults to None.

set_control_visibility(*layerControl=True*, *fullscreenControl=True*, *latLngPopup=True*)
Sets the visibility of the controls on the map.

Parameters

- **layerControl** (*bool, optional*) – Whether to show the control that allows the user to toggle layers on/off. Defaults to True.
- **fullscreenControl** (*bool, optional*) – Whether to show the control that allows the user to make the map full-screen. Defaults to True.

- **latLngPopup** (*bool, optional*) – Whether to show the control that pops up the Lat/lon when the user clicks on the map. Defaults to True.

set_options(*mapTypeId='HYBRID'*, *styles=None*, *types=None*)

Adds Google basemap and controls to the ipyleaflet map.

Parameters

- **mapTypeId** (*str, optional*) – A mapTypeId to set the basemap to. Can be one of "ROADMAP", "SATELLITE", "HYBRID" or "TERRAIN" to select one of the standard Google Maps API map types. Defaults to 'HYBRID'.
- **styles** (*object, optional*) – A dictionary of custom MapTypeStyle objects keyed with a name that will appear in the map's Map Type Controls. Defaults to None.
- **types** (*list, optional*) – A list of mapTypeIds to make available. If omitted, but opt_styles is specified, appends all of the style keys to the standard Google Maps API map types.. Defaults to None.

set_plot_options(*add_marker_cluster=False*, *sample_scale=None*, *plot_type=None*, *overlay=False*, *position='bottomright'*, *min_width=None*, *max_width=None*, *min_height=None*, *max_height=None*, ***kwargs*)

Sets plotting options.

Parameters

- **add_marker_cluster** (*bool, optional*) – Whether to add a marker cluster. Defaults to False.
- **sample_scale** (*float, optional*) – A nominal scale in meters of the projection to sample in . Defaults to None.
- **plot_type** (*str, optional*) – The plot type can be one of "None", "bar", "scatter" or "hist". Defaults to None.
- **overlay** (*bool, optional*) – Whether to overlay plotted lines on the figure. Defaults to False.
- **position** (*str, optional*) – Position of the control, can be 'bottomleft', 'bottomright', 'topleft', or 'topright'. Defaults to 'bottomright'.
- **min_width** (*int, optional*) – Min width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_width** (*int, optional*) – Max width of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **min_height** (*int, optional*) – Min height of the widget (in pixels), if None it will respect the content size. Defaults to None.
- **max_height** (*int, optional*) – Max height of the widget (in pixels), if None it will respect the content size. Defaults to None.

show_layer(*name*, *show=True*)

Shows or hides a layer on the map.

Parameters

- **name** (*str*) – Name of the layer to show/hide.
- **show** (*bool, optional*) – Whether to show or hide the layer. Defaults to True.

split_map(*left_layer='HYBRID'*, *right_layer='ROADMAP'*, *zoom_control=True*, *fullscreen_control=True*, *layer_control=True*, *add_close_button=False*, *left_label=None*, *right_label=None*, *left_position='bottomleft'*, *right_position='bottomright'*, *widget_layout=None*, ***kwargs*)

Adds split map.

Parameters

- **left_layer** (*str, optional*) – The layer tile layer. Defaults to 'HYBRID'.
- **right_layer** (*str, optional*) – The right tile layer. Defaults to 'ROADMAP'.
- **zoom_control** (*bool, optional*) – Whether to show the zoom control. Defaults to True.
- **fullscreen_control** (*bool, optional*) – Whether to show the full screen control. Defaults to True.
- **layer_control** (*bool, optional*) – Whether to show the layer control. Defaults to True.
- **add_close_button** (*bool, optional*) – Whether to add a close button. Defaults to False.
- **left_label** (*str, optional*) – The label for the left map. Defaults to None.
- **right_label** (*str, optional*) – The label for the right map. Defaults to None.
- **left_position** (*str, optional*) – The position of the left label. Defaults to 'bottomleft'.
- **right_position** (*str, optional*) – The position of the right label. Defaults to 'bottomright'.
- **widget_layout** (*str, optional*) – The layout of the label widget, such as ipywidgets.Layout(padding="0px 4px 0px 4px"). Defaults to None.
- **kwargs** – Other arguments for ipyleaflet.TileLayer.

to_gradio(*width='100%'*, *height='500px'*, ***kwargs*)

Converts the map to an HTML string that can be used in Gradio. Removes unsupported elements, such as attribution and any code blocks containing functions. See https://github.com/gradio-app/gradio/issues/3190

Parameters

- **width** (*str, optional*) – The width of the map. Defaults to '100%'.
- **height** (*str, optional*) – The height of the map. Defaults to '500px'.

Returns The HTML string to use in Gradio.

Return type str

to_html(*filename=None*, *title='My Map'*, *width='100%'*, *height='880px'*, *add_layer_control=True*, ***kwargs*)

Saves the map as an HTML file.

Parameters

- **filename** (*str, optional*) – The output file path to the HTML file.
- **title** (*str, optional*) – The title of the HTML file. Defaults to 'My Map'.

- **width** (*str, optional*) – The width of the map in pixels or percentage. Defaults to '100%'.
- **height** (*str, optional*) – The height of the map in pixels. Defaults to '880px'.
- **add_layer_control** (*bool, optional*) – Whether to add the LayersControl. Defaults to True.

to_image (*filename=None*, *monitor=1*)
Saves the map as a PNG or JPG image.

Parameters

- **filename** (*str, optional*) – The output file path to the image. Defaults to None.
- **monitor** (*int, optional*) – The monitor to take the screenshot. Defaults to 1.

to_streamlit (*width=None*, *height=600*, *scrolling=False*, ***kwargs*)
Renders map figure in a Streamlit app.

Parameters

- **width** (*int, optional*) – Width of the map. Defaults to None.
- **height** (*int, optional*) – Height of the map. Defaults to 600.
- **responsive** (*bool, optional*) – Whether to make the map responsive. Defaults to True.
- **scrolling** (*bool, optional*) – If True, show a scrollbar when the content is larger than the iframe. Otherwise, do not show a scrollbar. Defaults to False.

Returns components.html object.

Return type streamlit.components

toolbar_reset ()
Reset the toolbar so that no tool is selected.

ts_inspector (*left_ts*, *left_names=None*, *left_vis={}*, *left_index=0*, *right_ts=None*, *right_names=None*, *right_vis=None*, *right_index=-1*, *width='130px'*, *date_format='YYYY-MM-dd'*, *add_close_button=False*, ***kwargs*)
Creates a split-panel map for inspecting timeseries images.

Parameters

- **left_ts** (*object*) – An ee.ImageCollection to show on the left panel.
- **left_names** (*list*) – A list of names to show under the left dropdown.
- **left_vis** (*dict, optional*) – Visualization parameters for the left layer. Defaults to {}.
- **left_index** (*int, optional*) – The index of the left layer to show. Defaults to 0.
- **right_ts** (*object*) – An ee.ImageCollection to show on the right panel.
- **right_names** (*list*) – A list of names to show under the right dropdown.
- **right_vis** (*dict, optional*) – Visualization parameters for the right layer. Defaults to {}.
- **right_index** (*int, optional*) – The index of the right layer to show. Defaults to -1.
- **width** (*str, optional*) – The width of the dropdown list. Defaults to '130px'.

- **date_format** (*str, optional*) – The date format to show in the dropdown. Defaults to 'YYYY-MM-dd'.
- **add_close_button** (*bool, optional*) – Whether to show the close button. Defaults to False.

user_roi_coords(*decimals=4*)

Return the bounding box of the ROI as a list of coordinates.

Parameters **decimals** (*int, optional*) – Number of decimals to round the coordinates to. Defaults to 4.

video_overlay(*url*, *bounds*, *name='Video'*)

Overlays a video from the Internet on the map.

Parameters

- **url** (*str*) – http URL of the video, such as "https://www.mapbox.com/bites/00188/patricia_nasa.webm"
- **bounds** (*tuple*) – bounding box of the video in the format of (lower_left(lat, lon), upper_right(lat, lon)), such as ((13, -130), (32, -100)).
- **name** (*str*) – name of the layer to show on the layer control.

zoom_to_bounds(*bounds*)

Zooms to a bounding box in the form of [minx, miny, maxx, maxy].

Parameters **bounds** (*list | tuple*) – A list/tuple containing minx, miny, maxx, maxy values for the bounds.

zoom_to_gdf(*gdf*)

Zooms to the bounding box of a GeoPandas GeoDataFrame.

Parameters **gdf** (*GeoDataFrame*) – A GeoPandas GeoDataFrame.

zoom_to_me(*zoom=14*, *add_marker=True*)

Zoom to the current device location.

Parameters

- **zoom** (*int, optional*) – Zoom level. Defaults to 14.
- **add_marker** (*bool, optional*) – Whether to add a marker of the current device location. Defaults to True.

geemap.geemap.**ee_tile_layer**(*ee_object*, *vis_params={}*, *name='Layer untitled'*, *shown=True*, *opacity=1.0*)

Converts and Earth Engine layer to ipyleaflet TileLayer.

Parameters

- **ee_object** (*Collection|Feature|Image|MapId*) – The object to add to the map.
- **vis_params** (*dict, optional*) – The visualization parameters. Defaults to {}.
- **name** (*str, optional*) – The name of the layer. Defaults to 'Layer untitled'.
- **shown** (*bool, optional*) – A flag indicating whether the layer should be on by default. Defaults to True.
- **opacity** (*float, optional*) – The layer's opacity represented as a number between 0 and 1. Defaults to 1.

geemap.geemap.**linked_maps**(*rows=2*, *cols=2*, *height='400px'*, *ee_objects=[]*, *vis_params=[]*, *labels=[]*, *label_position='topright'*, ***kwargs*)

Create linked maps of Earth Engine data layers.

Parameters

- **rows** (*int, optional*) – The number of rows of maps to create. Defaults to 2.
- **cols** (*int, optional*) – The number of columns of maps to create. Defaults to 2.
- **height** (*str, optional*) – The height of each map in pixels. Defaults to "400px".
- **ee_objects** (*list, optional*) – The list of Earth Engine objects to use for each map. Defaults to [].
- **vis_params** (*list, optional*) – The list of visualization parameters to use for each map. Defaults to [].
- **labels** (*list, optional*) – The list of labels to show on the map. Defaults to [].
- **label_position** (*str, optional*) – The position of the label, can be [topleft, topright, bottomleft, bottomright]. Defaults to "topright".

Raises

- ValueError – If the length of ee_objects is not equal to rows*cols.
- ValueError – If the length of vis_params is not equal to rows*cols.
- ValueError – If the length of labels is not equal to rows*cols.

Returns A GridspecLayout widget.

Return type ipywidget

geemap.geemap.**ts_inspector**(*layers_dict=None*, *left_name=None*, *right_name=None*, *width='120px'*, *center=[40, -100]*, *zoom=4*, ***kwargs*)

4.1.7 geemap.legends module

Module of sample legends for some commonly used geospatial datasets.

geemap.legends.**ee_table_to_legend**(*in_table*, *out_file*)

Converts an Earth Engine color table to a dictionary

Parameters

- **in_table** (*str*) – The input file path (*.txt) to the Earth Engine color table.
- **out_file** (*str*) – The output file path (*.txt) to the legend dictionary.

4.1.8 Module contents

Top-level package for geemap.

geemap.**use_folium**()

Whether to use the folium or ipyleaflet plotting backend.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/giswqs/geemap/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

geemap could always use more documentation, whether as part of the official geemap docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/giswqs/geemap/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *geemap* for local development.

1. Fork the *geemap* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/geemap.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv geemap
$ cd geemap/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 geemap tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/giswqs/geemap/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_geemap
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Qiusheng Wu

6.2 Contributors

- Cesar Aybar
- Oliver Burdekin
- Diego Garcia Diaz
- Justin Braaten

CHAPTER 7

History

7.1 0.7.0 (2020-05-22)

7.2 0.6.0 (2020-04-05)

7.3 0.5.0 (2020-03-24)

7.4 0.4.0 (2020-03-19)

7.5 0.3.0 (2020-03-18)

7.6 0.2.0 (2020-03-17)

7.7 0.1.0 (2020-03-08)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

A

B

C

D

E

F

G

I

J

L

M

P

Q

R

S

T

U

V

X

Z